Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Análisis de Vulnerabilidades en Bluetooth

Bluetooth vlunerability Anlysis

La Laguna, 14 de julio de 2023

D. **Mª Candelaria Hernández Goya**, con N.I.F. 45.441.714-Q profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de sistemas de la Universidad de La Laguna, como tutor

D. **José Iván Santos González**, con N.I.F. 78.637.989-T como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"Análisis de Vulnerabilidades en Bluetooth"

ha sido realizada bajo su dirección por D. **Milton Daniel Rivas Quintero**,

con N.I.F. 42.268.545-L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2023

Agradecimientos

En primer lugar, me gustaría agradecer todo el trabajo y apoyo recibido a mis tutores María y José y al resto de profesores que me guiaron a lo largo de la carrera.

A mi madre por ser mi pilar y mi ejemplo para seguir cada día.

A mi hermana por ser un apoyo incondicional y estar siempre conmigo en cada momento.

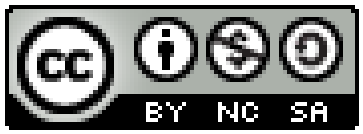
A mi abuela por el cariño y la alegría que me ha dado.

Al resto de mi familia porque siempre me apoyan en cada momento y me sacan una sonrisa hasta en los momentos más duros

A mis amigos (mis panas) que nos apoyamos mutuamente a lo largo de la carrera.

A la Universidad de La Laguna por ser mi Alma mater.

Licencia



Resumen

En un mundo cada vez más conectado, la seguridad de las comunicaciones es de vital importancia para proteger la confidencialidad e integridad de los datos transmitidos. La tecnología Bluetooth, como una de las formas más comunes de comunicación inalámbrica, requiere mecanismos sólidos de seguridad para salvaguardar la información personal y sensible.

Una de las tecnologías inalámbricas de comunicación más extendidas es Bluetooth y por esta razón es necesario aplicar mecanismo de seguridad para evitar cualquier posible robo de nuestra información de los datos mediante esta tecnología.

El objetivo de este trabajo fue desarrollar una aplicación móvil que permitiera transferir datos de manera segura a través de Bluetooth, implementando capas adicionales de seguridad para garantizar la integridad de los datos transferidos. Se llevó a cabo una investigación exhaustiva sobre las tecnologías y conceptos involucrados, como el desarrollo de aplicaciones en Android, el funcionamiento de Bluetooth y las primitivas criptográficas.

Palabras clave: Bluetooth, Android, Aplicación móvil, Usabilidad, Accesibilidad, Cifrado, Diffie-Hellman

Abstract

In an increasingly connected world, the security of communications is of vital importance to protect the confidentiality and integrity of transmitted data. Bluetooth technology, as one of the most common forms of wireless communication, requires robust security mechanisms to safeguard personal and sensitive information.

One of the most widespread wireless communication technologies is Bluetooth, and for this reason, it is necessary to apply security mechanisms to prevent any possible theft of our data through this technology.

The objective of this work was to develop a mobile application that allows for secure data transfer over Bluetooth, implementing additional layers of security to ensure the integrity of the transferred data. An exhaustive investigation was carried out on the involved technologies and concepts, such as Android application development, the functioning of Bluetooth, and cryptographic primitives.

Keywords: Bluetooth, Android, Mobile application, Usability, Accessibility, Encryption, Diffie-Hellman

Índice general

Contenido

Índice de figuras

Índice de tablas

Capítulo 1 Introducción

1.1 Motivación

Las redes de área personal, más conocidas por sus siglas en inglés PAN (Personal Area Networks), son un tipo de red de dispositivos que se encuentran a una distancia máxima de alcance de 10 metros aproximadamente, y que permiten el intercambio de datos entre ellos. Además, existen las redes inalámbricas de Área personal WPAN (Wireless Personal Area Networks) donde los dispositivos se conectan mediante ondas electromagnéticas sin necesidad de cables.

Una de las tecnologías más destacadas dentro de WPAN es Bluetooth, lanzada por Ericsson en 1994. Se perseguía desarrollar una interfaz abierta de fácil implementación y uso, con bajo consumo energético y baja latencia para facilitar la comunicación entre dispositivos sin la utilización de cables, aprovechando la movilidad de los dispositivos inalámbricos.

Conociendo estas ventajas, hay que tener en cuenta los riesgos que se añaden [1], además de los que ya existen en las redes cableadas (ataques person-in-the-middle, ataques de denegación de servicio, eavesdropping, MAC spoofing, etc.). Estos riesgos deben tratarse de forma específica, e implementar las medidas necesarias para proporcionar seguridad en la comunicación que se lleva a cabo mediante esta tecnología.

La conexión en Bluetooth no es del todo segura, es verdad que poco a poco se encuentran más smartphones con Bluetooth 5.0, pero la mayoría de los presentes en el mercado cuentan con Bluetooth 4.1 o versiones más antiguas. Desde la versión 4.2, la conectividad Bluetooth cuenta con una nueva aproximación en la implementación de servicios de seguridad (SSP y AES-CCM), pero contiene también vulnerabilidades. Estas especificaciones de Bluetooth cuentan con cuatro métodos de emparejamiento diferentes, conocidos como Numeric Comparison, Just Works, Out–of-Band y Passkey Entry. Cada uno de ellos tiene sus propios defectos. Just Works es vulnerable a numerosos ataques y Out–of-Band requiere un canal separado para la comunicación (no todos los dispositivos lo admiten). En el caso de Passkey Entry se puede interceptar por un tercer interlocutor. Por estas

razones, en este Trabajo Final de Grado se plantea realizar una aplicación que permita al usuario definir su propio conjunto de servicios de seguridad para proteger la información intercambiada a través del canal Bluetooth.

1.2 Antecedentes y estado actual del tema

Con el paso del tiempo se han encontrado varias vulnerabilidades en Bluetooth, lo que demuestra que es una tecnología en evolución y que es aconsejable que el usuario cuente con medidas de protección adicionales a la hora de usarla. A continuación, se enumeran algunas de las vulnerabilidades más recientes.

- **Múltiples vulnerabilidades en dispositivos que soportan especificaciones Bluetooth** [2][3]: investigadores de la ANSSI (Agence nationale de la sécurité des systèmes d'information) han descubierto una serie de vulnerabilidades, a las que se les ha dado el nombre de BIAS (Bluetooth Impersonation AttackS), en las especificaciones Core[4] y Mesh Profile[**5**] del protocolo Bluetooth que podrían permitir a un atacante suplantar la identidad de un dispositivo legítimo.
- **Amazon resuelve una vulnerabilidad de los Echo que evita que se "auto hackeen"** [6]: Alexa versus Alexa (AvA), es un ataque que aprovecha los archivos de audio que contienen comandos de voz y métodos de reproducción de audio de manera ofensiva para obtener el control de los dispositivos Amazon Echo durante un período de tiempo prolongado [7].
- **BrakTooth, la vulnerabilidad de 'bluetooth' que puede bloquear millones de dispositivos** [8]: la vulnerabilidad correspondiente con el CVE-2021-28139 y afecta a los SoC (System-on-Chip) ESP32[9]. Se basa en que un atacante se hace con el control de un dispositivo al ejecutar su propio código corrupto en los dispositivos afectados a través de paquetes LMP (Link Manager Protocol) que se usa para el establecimiento y control del enlace de radio entre los dos dispositivos [10].

Al consultar la base de datos de vulnerabilidades cvedetails [11] se obtiene un amplio conjunto referente a la tecnología Bluetooth. A modo de ejemplo se relacionan a continuación algunas de las más significativas:

- **Bluetooth Classic en Espressif** [12]: un atacante se hace con el control de un dispositivo al ejecutar su propio código corrupto en los dispositivos afectados a través de paquetes LMP

(Link Manager Protocol) que se utiliza para el establecimiento y control del enlace de radio entre los dos dispositivos [10].

- **Protocolo Passkey Entry** [13]: un atacante que actúe como intermediario (Person-in-the-Middle) en el procedimiento de autenticación de la clave de acceso podría usar una serie de respuestas diseñadas para determinar cada bit de la clave de acceso generada aleatoriamente seleccionada por el iniciador del emparejamiento en cada ronda del procedimiento de emparejamiento, y una vez identificado, el atacante puede usar estos bits de clave de acceso durante la misma sesión de emparejamiento para completar con éxito el procedimiento de emparejamiento autenticado con el respondedor.
- **Bluetooth Mesh Profile AuthValue leak** [14]: el proceso de Mesh Provisioning podría permitir a un atacante, que fue aprovisionado sin acceso al AuthValue, identificarlo directamente sin necesidad de realizar un ataque de fuerza bruta para averiguar su valor.
- **Impersonation attack in Bluetooth Mesh Profile provisioning** [15]: el procedimiento de Mesh Provisioning podría permitir a un atacante, sin conocimiento del AuthValue, realizar un spoofing a un dispositivo que se está aprovisionando, utilizar respuestas especialmente diseñadas para simular que posee el AuthValue y recibir una NetKey válida y potencialmente una AppKey.
- **Impersonation in the BR/EDR pin-pairing protocol** [16]: un atacante podría conectarse al dispositivo de una víctima realizando un spoofing del BD_ADDR (Bluetooth Device Address) para suplantar el proceso de Bluetooth BR/EDR PIN Pairing.
- **Bluetooth Impersonation Attacks** [17]: el emparejamiento heredado y la autenticación de emparejamiento de conexiones seguras en Bluetooth BR/EDR Core Specification v5.2 y anteriores pueden permitir que un usuario no autenticado complete la autenticación sin credenciales de emparejamiento a través de un acceso adyacente. Un atacante adyacente no autenticado podría hacerse pasar por un maestro o esclavo BR/EDR de Bluetooth para emparejarse con un dispositivo remoto previamente emparejado para completar con éxito el procedimiento de autenticación sin conocer la clave de enlace.
- **Vulnerabilidad de fuga de información** [18]: una vulnerabilidad clasificada como obtención de información fue encontrada en la respuesta de escaneo de anuncios de Bluetooth Low Energy en las especificaciones principales de Bluetooth 4.0 a 5.2 y la respuesta de escaneo extendida en las especificaciones principales de Bluetooth 5.0 a 5.2 se pueden usar para identificar dispositivos que usan Direccionamiento privado resoluble (RPA) por su respuesta o no. La vulnerabilidad fue publicada el 2022-11-08. La vulnerabilidad es identificada como

CVE-2020-35473. No se conocen los detalles técnicos ni hay ningún exploit disponible.

En relación con el estado actual de esta tecnología y de las soluciones que las despliegan, se deben destacar algunas aplicaciones relacionadas con la que se desarrollará en este Trabajo Final de Grado:

Bluetooth File Transfer [19]: esta aplicación permite buscar, explorar y gestionar archivos de cualquier dispositivo Bluetooth, usando el Perfil de Transferencia de Archivos (FTP) y el Perfil de Empuje de Objetos (OPP): se podría también recibir archivos y enviar contactos.

Transfer archivos Bluetooth [20]: con esta aplicación se pueden compartir diferentes tipos de archivos, desde fotos hasta música o documentos. Además, ofrece una serie de características que facilitan su uso y mejoran la experiencia de usuario, como lo son: compartir contactos, copia de seguridad de la aplicación, interruptor de Bluetooth propio, etc.

Compartir aplicación [21]: esta es una aplicación muy sencilla donde su principal función es compartir otras aplicaciones transfiriendo el archivo .apk de la misma.

Sin embargo, ninguna de ellas implementa servicios de seguridad adicionales.

1.3 Objetivos

Actualmente vivimos en un mundo tan globalizado y donde los teléfonos celulares se convirtieron en aparatos tan imprescindibles que es difícil pensar vivir sin uno. Esto conlleva a que cada vez estemos más conectados y compartiendo información por distintos métodos.

El objetivo del trabajo es entender cómo funciona Bluetooth [22], una de las principales tecnologías que nos permiten estos procesos de intercambio de información, conocer sus vulnerabilidades, y de esta forma, ver cómo proteger nuestra información de posibles atacantes.

Se desarrollará una aplicación móvil, para el sistema operativo Android [23], utilizando el lenguaje de programación Kotlin [24], donde su cometido será la transferencia de datos entre teléfonos móviles usando el estándar de conectividad inalámbrica Bluetooth, pero agregando características de seguridad para dificultar el acceso a los datos por parte de un atacante.

1.4 Estado del arte

Bluetooth es un estándar de comunicación inalámbrico de corto alcance que permite la conexión de dispositivos electrónicos. Fue desarrollado en 1994 por Ericsson y desde entonces ha evolucionado

a través de varias versiones. La última versión estable es Bluetooth 5.4, lanzada en 2023 [25].

La última versión de Bluetooth ofrece una serie de mejoras respecto a sus predecesores, algunas de estas características son:

- Comunicación bidireccional con miles de nodos finales desde un solo punto de acceso: Bluetooth 5.4 permite establecer una comunicación bidireccional eficiente con un gran número de dispositivos finales desde un único punto de acceso. Esto es especialmente útil en aplicaciones de Internet de las cosas (IoT) que requieren una conectividad escalable.
- Publicidad Periódica con Respuestas (PAwR): Esta nueva característica permite la implementación de una red estrella con sincronización de tiempo y comunicación bidireccional mejorada. Los dispositivos pueden recibir datos de un anunciante periódico y responder al transmisor del anunciante. También se pueden asignar dispositivos a grupos específicos para escuchar solo las transmisiones de su grupo.
- Datos de Publicidad Encriptados (EAD): Bluetooth 5.4 introduce la capacidad de encriptar los datos de publicidad transmitidos. Estos datos encriptados solo pueden ser descifrados y autenticados por dispositivos que hayan compartido previamente la clave de sesión. Esto mejora la seguridad y privacidad de la comunicación en entornos Bluetooth.

Bluetooth es el resultado de muchos años de desarrollo y mejora continua, y se ha convertido en un estándar muy importante para la conexión de dispositivos electrónicos. Se espera que en el futuro siga evolucionando y se utilice en una amplia gama de aplicaciones y principalmente en IoT.

1.5 Fases y desarrollo del proyecto

El objetivo principal de este trabajo es desarrollar una aplicación de Android [26] que permita a los usuarios enviar archivos entre dispositivos mediante Bluetooth, incorporando una capa adicional de seguridad mediante el cifrado de los datos enviados y su posterior descifrado en el dispositivo receptor. Para lograr este objetivo, fue necesario realizar un estudio previo de diversas tecnologías y conceptos que previamente desconocía. Este estudio abarcó desde el desarrollo móvil en Android hasta la utilización de librerías específicas de cifrado para lenguajes de programación como Kotlin [24] y Java [28]. El propósito de esta investigación fue garantizar la confidencialidad y la integridad de los archivos transmitidos, así como prevenir posibles vulnerabilidades en el proceso de transferencia de datos a través de la conexión Bluetooth.

1.5.1 Estudio preliminar de tecnologías y conceptos relacionados

Durante la fase inicial del proyecto, se realizó un estudio preliminar de diversas tecnologías y conceptos clave relacionados con el desarrollo de la aplicación. A continuación, se detallan las principales áreas de investigación:

- Desarrollo móvil en Android: Se llevó a cabo un análisis de la plataforma Android, incluyendo el estudio de su arquitectura, componentes y ciclo de vida de las actividades. Se investigaron buenas prácticas y patrones de diseño recomendados para el desarrollo de aplicaciones Android, con el objetivo de garantizar una estructura sólida y un rendimiento óptimo.
- API de Bluetooth de Android: Se exploró en la API de Bluetooth de Android para comprender su funcionamiento y las capacidades que ofrece.
- Lenguajes de programación Kotlin [24]: Se hicieron pequeñas aplicaciones para entender cómo funciona Kotlin y ver las características y ventajas que ofrece.
- Android Studio [29]: El proyecto comenzó desarrollándose con la versión de Electric Eel de Android Studio el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android, pero se finalizó usando la versión Flamingo, se hicieron varias pruebas de aplicaciones simples para familiarizarse con este IDE y comprobar las opciones y características que ofrece, como, por ejemplo: Interfaz intuitiva, Editor de código inteligente, Consola de depuración o Emulador y dispositivos virtuales, estas y otras características se detallan en el siguiente Capítulo 2 Herramientas de desarrollo.

Capítulo 2 Herramientas de desarrollo

2.1 Android Studio

Android Studio [29] es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) creado específicamente para el desarrollo de aplicaciones móviles en el sistema operativo Android. Proporciona a los desarrolladores una amplia gama de herramientas y funciones para crear, depurar y optimizar aplicaciones de manera eficiente. Se basa en el IDE IntelliJ IDEA de JetBrains y está optimizado para el desarrollo de aplicaciones Android. Está escrito principalmente en Java y utiliza el lenguaje de programación Kotlin como un enfoque preferido en lugar de Java para el desarrollo de aplicaciones Android.

Android Studio ofrece una amplia variedad de características y funcionalidades para facilitar el desarrollo de aplicaciones, por ejemplo, algunas de las características clave son:

- Editor de código: Android Studio proporciona un editor de código altamente funcional con resaltado de sintaxis, finalización automática y verificación de errores en tiempo real. Admite múltiples lenguajes, como Java y Kotlin, y ofrece herramientas para refactorizar el código y mejorar su calidad.

- Interfaz intuitiva: Android Studio ofrece una interfaz de usuario intuitiva y fácil de usar a mi parecer, diseñada específicamente para el desarrollo de aplicaciones Android. Proporciona un entorno de trabajo organizado y visualmente agradable, con paneles y herramientas bien distribuidas para facilitar la navegación y el acceso rápido a las funciones clave, en la Figura 1 se puede observar cómo es la interfaz de Android Studio Flamingo.

- Administrador de proyectos: Permite crear y administrar proyectos de Android de manera eficiente. Facilita la organización de los archivos de la aplicación y proporciona herramientas para importar, exportar y compartir proyectos, en la Figura 2 Estructura inicial del proyecto se puede apreciar los directorios iniciales que se crean cuando empiezas un proyecto usando Android Studio.
- Integración con GitHub: Android Studio ofrece integración con GitHub, lo que permite a los desarrolladores gestionar y colaborar en proyectos utilizando el control de versiones Git. Esto facilita el trabajo en equipo y la gestión de cambios en el código fuente de la aplicación.

- Depuración y pruebas: Proporciona herramientas robustas para la depuración de aplicaciones, como puntos de interrupción, inspección de variables y seguimiento de pila. Además, permite ejecutar pruebas unitarias y pruebas de interfaz de usuario para garantizar la calidad y el rendimiento de la aplicación.

- Emulador y dispositivos virtuales: Android Studio incluye un emulador de Android que permite probar aplicaciones en diferentes dispositivos virtuales con diferentes versiones de Android. Esto ayuda a asegurarse de que la aplicación funcione correctamente en una amplia gama de dispositivos antes de la implementación. También es posible conectar dispositivos físicos para probar la aplicación directamente en hardware real, que esto último fue la opción que use porque los dispositivos emulados no cuentan con el servicio de Bluetooth [30].

- Integración de herramientas externas: Android Studio se integra con otras herramientas populares, como el SDK de Android, Gradle (sistema de compilación), Firebase (plataforma de desarrollo móvil de Google), entre otras. Esto facilita el acceso a funciones adicionales y la integración de servicios en las aplicaciones.

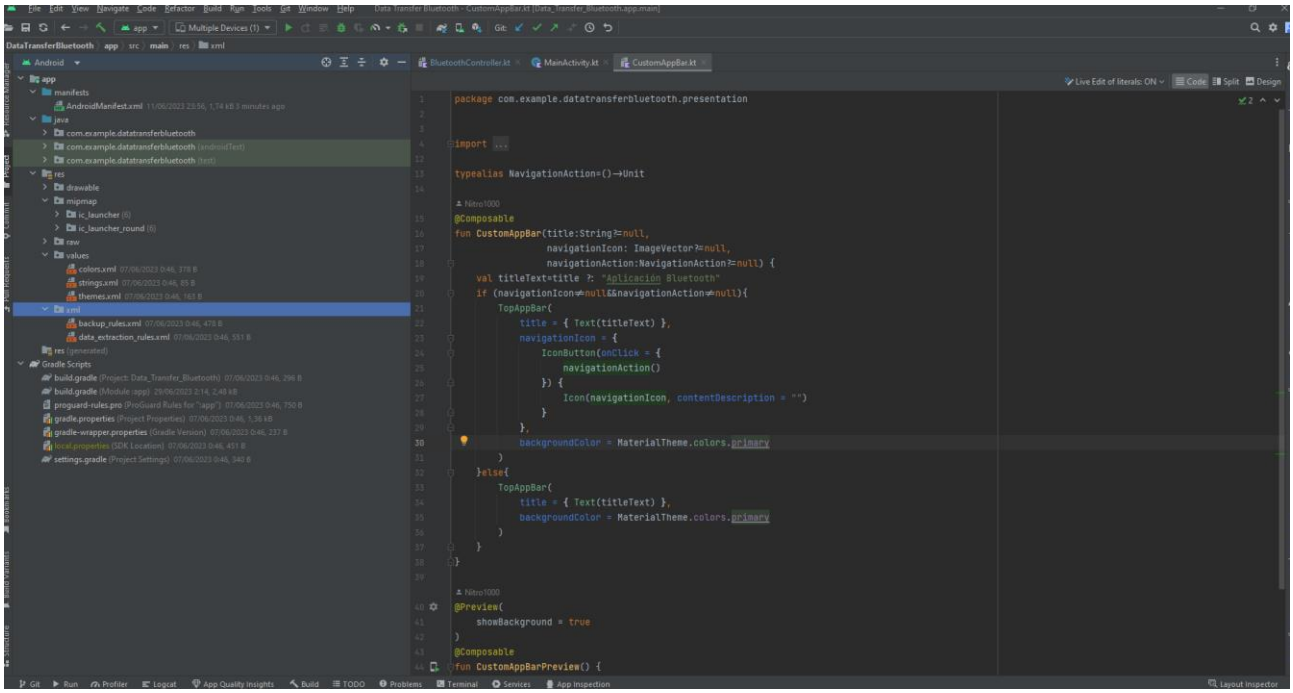


Figura 1 Interfaz de Android Studio

2.2 Figma

Figma [31] es una herramienta de diseño y prototipado de interfaces de usuario (UI) basada en la nube. Se utiliza ampliamente en el ámbito del diseño de productos digitales y la colaboración entre equipos de diseño y desarrollo. Figma ofrece un enfoque colaborativo en tiempo real que permite a los diseñadores crear y compartir diseños, iterar sobre ellos y recopilar comentarios de manera eficiente.

Figma ofrece un conjunto de herramientas poderosas para diseñar interfaces de usuario. Los diseñadores pueden crear y editar elementos gráficos como formas, iconos, imágenes y textos directamente en la interfaz de Figma. Además, proporciona una amplia gama de funciones de diseño, como capas, estilos de texto, máscaras, efectos de sombra y degradados, lo que permite crear diseños visualmente atractivos y funcionales.

Una de las características más destacadas de Figma es su capacidad para permitir la colaboración en tiempo real. Múltiples diseñadores pueden trabajar en un mismo archivo simultáneamente, lo que facilita la comunicación y la iteración rápida entre los miembros del equipo. Además, Figma ofrece la posibilidad de comentar y realizar anotaciones en los diseños, lo que fomenta una comunicación efectiva y una retroalimentación precisa.

Otra ventaja de Figma es su funcionalidad de prototipado. Los diseñadores pueden crear enlaces interactivos entre diferentes pantallas y definir transiciones, animaciones y estados de componentes para simular la experiencia del usuario final. Esto permite validar y probar la usabilidad del diseño antes de pasar a la etapa de desarrollo.

2.3 Git

Git [32] es un sistema de control de versiones distribuido ampliamente utilizado en el desarrollo de software. Proporciona una forma eficiente de rastrear y administrar los cambios realizados en los archivos de un proyecto a lo largo del tiempo. Git utiliza una estructura de datos llamada "árbol de versiones" para almacenar y organizar los cambios realizados en los archivos. En Git, un repositorio es el contenedor principal que almacena todos los archivos y la historia de cambios de un proyecto. Cada vez que se realiza una modificación en los archivos del proyecto, Git registra estos cambios en forma de "commits". Un commit en Git representa un conjunto lógico de cambios y se identifica por un identificador único. Cada commit contiene una referencia a su commit padre, lo que crea una

estructura de grafo dirigido que representa la historia de cambios del proyecto.

Git es una herramienta distribuida, lo que significa que cada colaborador tiene una copia completa del repositorio en su propio sistema. Esto permite a los desarrolladores trabajar de forma aislada y sin conexión a Internet, y luego fusionar sus cambios con el repositorio principal en línea.

Además de su capacidad de rastrear y gestionar cambios, Git ofrece una variedad de funciones que facilitan el desarrollo colaborativo y el trabajo en equipo. Algunas de las características clave de Git incluyen:

- Ramificación y fusión: Git permite crear ramas separadas del desarrollo principal para trabajar en nuevas características o solucionar problemas sin afectar el código base. Posteriormente, las ramas se pueden fusionar para combinar los cambios.
- Etiquetas: Git permite etiquetar commits específicos para marcar hitos importantes en el desarrollo, como versiones de lanzamiento o puntos de referencia.
- Resolución de conflictos: Cuando dos o más desarrolladores realizan cambios en la misma parte de un archivo, puede haber conflictos. Git proporciona herramientas para resolver estos conflictos de manera eficiente y colaborativa.
- Repositorios remotos: Git permite la colaboración en proyectos a través de repositorios remotos. Los desarrolladores pueden clonar repositorios remotos en sus sistemas locales, trabajar en ellos y luego sincronizar sus cambios con el repositorio remoto compartido.

2.4 GitHub

GitHub [33] es una plataforma de desarrollo colaborativo basada en la nube que utiliza el sistema de control de versiones Git. Se utiliza ampliamente en el ámbito de la programación y el desarrollo de software para gestionar y compartir proyectos de forma eficiente.

En GitHub, puedes crear repositorios que contienen los archivos de tu proyecto, incluyendo el código fuente, la documentación y otros recursos. Estos repositorios actúan como una versión centralizada de tu proyecto, lo que te permite realizar un seguimiento de los cambios, colaborar con otros desarrolladores y mantener un historial completo de las modificaciones realizadas. Además de su función principal como plataforma de alojamiento de repositorios Git, GitHub también ofrece características adicionales, como la gestión de problemas (issues), que permite realizar un seguimiento de los problemas, errores o tareas pendientes en un proyecto. También cuenta con un sistema de integración continua (CI) que te permite automatizar pruebas y despliegues para garantizar

la calidad del código.

2.5 Excalidraw

Excalidraw [34] es una herramienta de dibujo en línea que se utiliza para crear diagramas y bocetos colaborativos. Está diseñada para ser simple, intuitiva y fácil de usar, lo que la hace ideal para la creación de diagramas rápidos y esbozos conceptuales.

Excalidraw permite la colaboración en tiempo real, lo que significa que varias personas pueden trabajar en un mismo dibujo simultáneamente. Puedes invitar a otros usuarios a unirse a tu sesión de dibujo compartiendo un enlace, y verás los cambios que realizan en tiempo real. Esto facilita la colaboración remota y la comunicación visual entre miembros de un equipo.

Además, Excalidraw permite exportar tus dibujos en diferentes formatos, como imágenes PNG o SVG, lo que te permite compartirlos fácilmente con otros o incorporarlos en tu documentación o presentaciones.

2.6 Test de accesibilidad

Test de accesibilidad de Google [35] permite analizar la interfaz de usuario de una aplicación y brindar recomendaciones para mejorar su accesibilidad. Esta herramienta es útil tanto para desarrolladores como para usuarios, ya que proporciona una forma rápida y sencilla de identificar una amplia gama de mejoras de accesibilidad.

Al utilizar la Prueba de Accesibilidad, es posible detectar y corregir problemas comunes de accesibilidad, como el tamaño insuficiente de los objetivos táctiles, el contraste deficiente en el texto y las imágenes, y la falta de descripciones adecuadas para los elementos gráficos sin etiquetar. Estas recomendaciones permiten mejorar la experiencia de uso para todas las personas, incluyendo aquellas con discapacidades visuales, motoras o cognitivas.

La herramienta analiza la aplicación en tiempo real y proporciona retroalimentación inmediata sobre los aspectos de accesibilidad que podrían mejorarse. Esto incluye sugerencias sobre cambios en el diseño, ajustes en los colores utilizados, mejoras en la legibilidad del texto y mucho más.

Al realizar pruebas de accesibilidad utilizando esta herramienta, se asegura que la aplicación cumpla con los estándares de accesibilidad y proporcione una experiencia inclusiva para todos los usuarios. Esto es especialmente importante para garantizar que las personas con discapacidades

puedan acceder y utilizar la aplicación de manera efectiva. Para utilizar la herramienta seguí los pasos de la guía que ofrece Google [36]

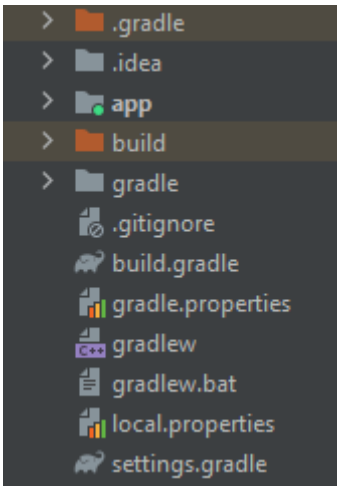


Figura 2 Estructura inicial del proyecto

Capítulo 3 Diagramas

3.1 Diagramas de caso de uso

El diagrama de casos de uso es una representación gráfica que describe las interacciones entre los actores (usuarios o sistemas externos) y el sistema en estudio. Este diagrama se utiliza para identificar y visualizar los diferentes casos de uso o funcionalidades del sistema y cómo se relacionan con los actores. En un diagrama de casos de uso, se muestran los actores, los casos de uso y las relaciones entre ellos mediante líneas que representan las interacciones.

Cada caso de uso representa una función o acción que puede realizar un actor en el sistema. Los actores son los roles que interactúan con el sistema, como usuarios finales, administradores u otros sistemas externos. Los casos de uso se representan como elipses y los actores como figuras más grandes. Las relaciones entre actores y casos de uso se muestran mediante flechas que indican las interacciones. En la Figura 3 Casos de uso se pueden ver como interaccionan los usuarios con la aplicación.

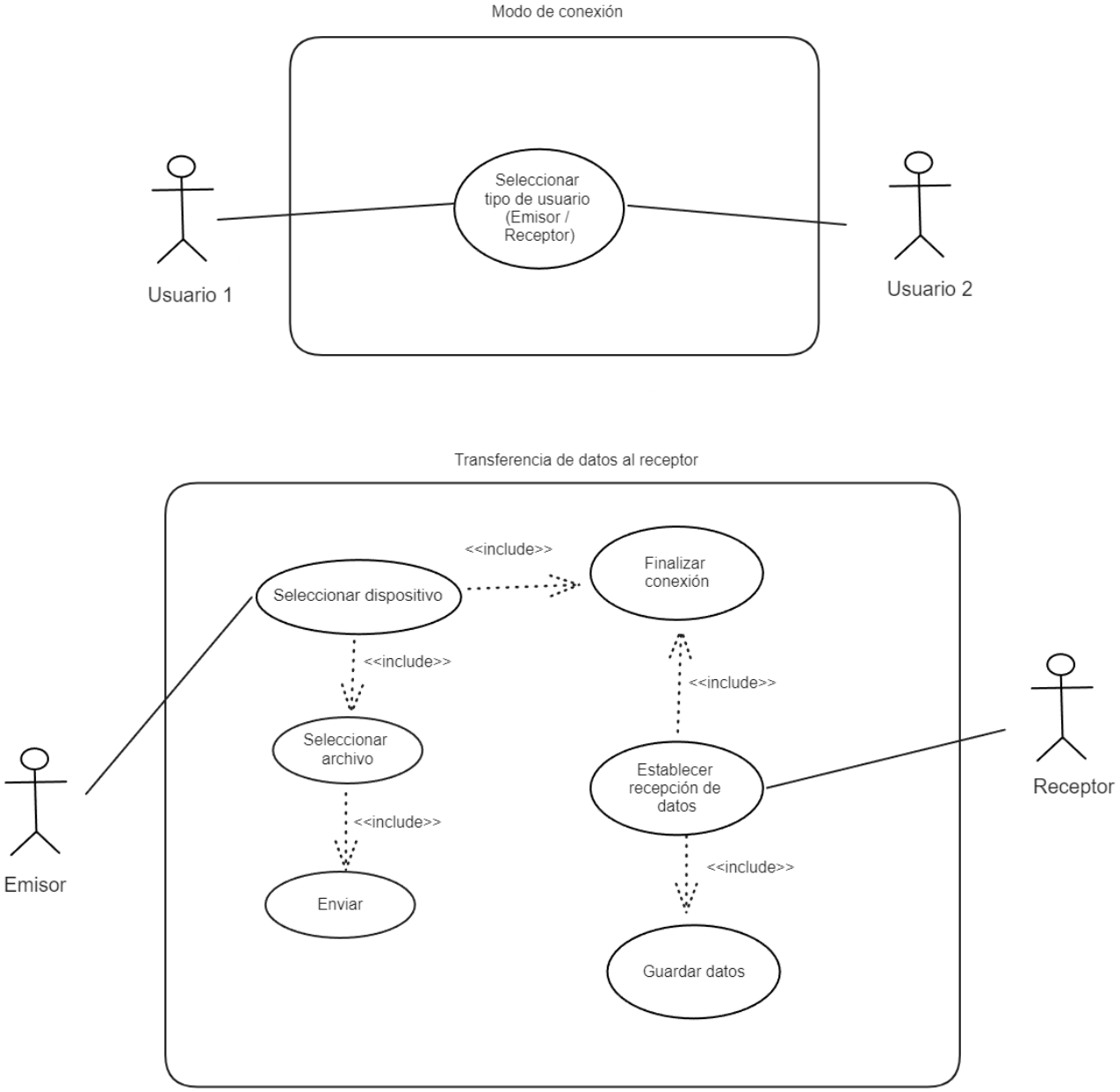


Figura 3 Diagrama de casos de uso

3.2 Diagrama de clases

El diagrama de clases es una representación visual de las clases, sus atributos y las relaciones entre ellas en un sistema orientado a objetos. Este diagrama se utiliza para modelar la estructura estática del sistema y cómo se organizan las clases y los objetos en él.

En un diagrama de clases, las clases se representan como rectángulos divididos en tres secciones: la sección superior contiene el nombre de la clase, la sección media muestra los atributos de la clase y la sección inferior muestra los métodos o funciones de la clase. Las relaciones entre las clases se representan mediante líneas que indican la asociación, la herencia, la dependencia u otras relaciones entre ellas. En la Figura 4 Diagrama de clase, se puede ver como se estructura internamente el código de la aplicación.

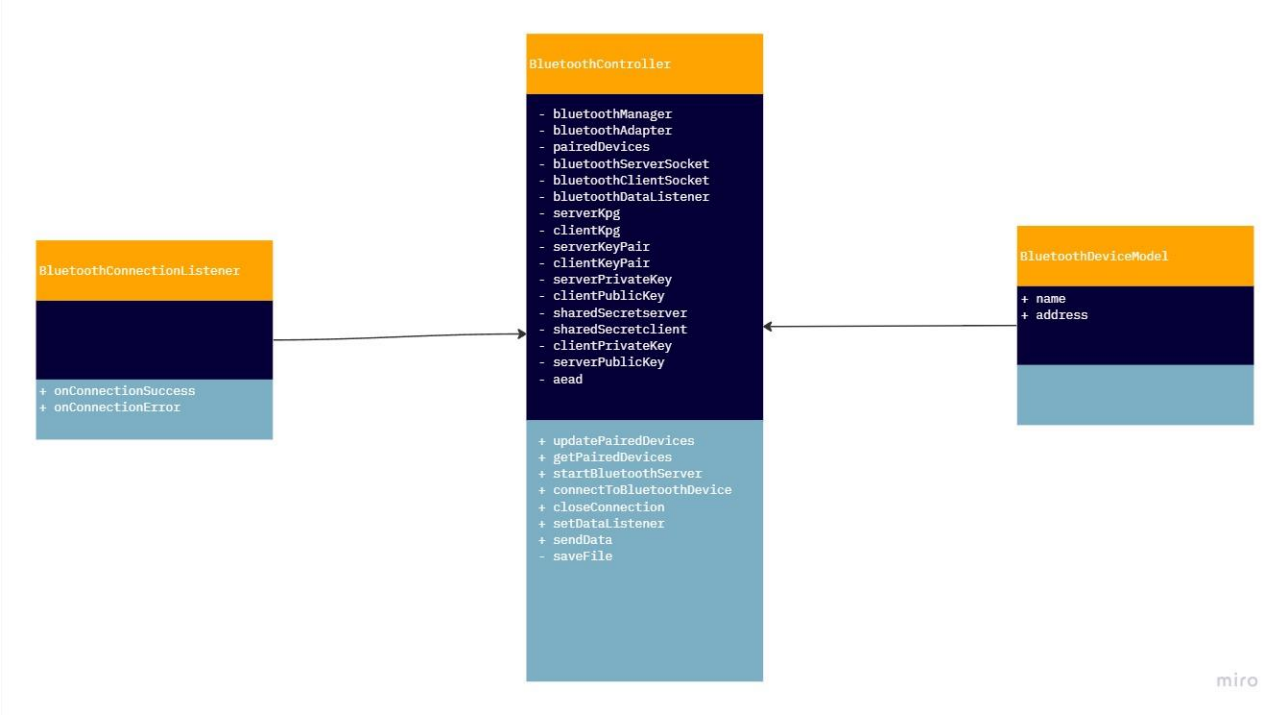


Figura 4 Diagrama de clase

Capítulo 4 Descripción de BlueShare

Como se comento en el apartado 1.5 Fases y desarrollo del proyecto el objetivo principal de este trabajo es desarrollar una aplicación de Android [26] que permita a los usuarios enviar archivos entre dispositivos mediante Bluetooth, incorporando una capa adicional de seguridad mediante el cifrado de los datos, por ende, quería que el nombre de la aplicación representara su función principal, que es la de compartir datos a través de Bluetooth y se me ocurrió llamarla “BlueShare”.

4.1 Arquitectura de la aplicación

Para el desarrollo de la aplicación, se ha adoptado una arquitectura basada en el patrón Modelo-Vista-Controlador (MVC) [36], en la Figura 5 Modelo-Vista-Controlador se puede ver un esquema en ejemplifica este patrón de diseño. En esta arquitectura, se ha diseñado un controlador Bluetooth como una clase central que se encarga de implementar toda la lógica relacionada con las funcionalidades basadas en Bluetooth. El controlador Bluetooth actúa como un intermediario entre las vistas y las operaciones de Bluetooth.

Las vistas se han implementado utilizando Jetpack Compose [37], un conjunto de herramientas de Android desarrolladas por Google para construir interfaces de usuario de manera declarativa, lo que permite describir la interfaz de usuario especificando qué aspecto y comportamiento debería tener, en lugar de programar manualmente los pasos detallados para crearla.

En un enfoque declarativo, en lugar de escribir código detallado que indique cómo se debe construir y actualizar la interfaz de usuario paso a paso, se describe el resultado deseado de la interfaz de usuario en términos más abstractos. Se proporciona una descripción de alto nivel de los componentes, su estructura y cómo se relacionan entre sí.

A diferencia de los enfoques tradicionales basados en XML en el caso de Jetpack Compose, se utiliza Kotlin [24] como lenguaje de programación para describir la interfaz de usuario de manera declarativa. En lugar de crear y manipular objetos de vista de forma explícita, se utilizan funciones llamadas "composables" para definir los elementos de la interfaz de usuario.

Un composable representa un componente visual, como un botón, un cuadro de texto o una lista, y se describe cómo debería verse y comportarse. En lugar de especificar los pasos para crear manualmente el botón, establecer su color, tamaño y escuchar eventos de clic, se define el composable

del botón y se especifican los atributos como color, tamaño y eventos de clic directamente en la descripción del composable.

Esta aproximación declarativa permite una mayor simplicidad y claridad en la implementación de la interfaz de usuario. Además, facilita la comprensión del código y la realización de cambios, ya que solo es necesario modificar la descripción del composable para reflejar los cambios deseados en la interfaz de usuario.

Las vistas (Composables), interactúan con el controlador Bluetooth mediante el uso de los métodos expuestos por esta clase. Al acceder a los métodos del controlador Bluetooth, las vistas pueden realizar diversas acciones basadas en las solicitudes del usuario.

La elección de la arquitectura Modelo-Vista-Controlador (MVC) ha brindado una estructura organizada para el código de la aplicación. La separación clara de responsabilidades permite una mejor mantenibilidad y escalabilidad del código, facilitando la implementación de nuevas características y la gestión de cambios.

Al seguir esta arquitectura, se ha logrado una implementación modular y flexible de la funcionalidad Bluetooth en la aplicación. El controlador Bluetooth se encarga de establecer la conexión con otros dispositivos Bluetooth, ya sea como servidor o como cliente, y maneja el proceso de transferencia de archivos de manera segura. Además, se ha tenido en cuenta la seguridad y se ha incorporado un cifrado de datos para garantizar la confidencialidad durante la transferencia, que se explicara en el siguiente apartado.

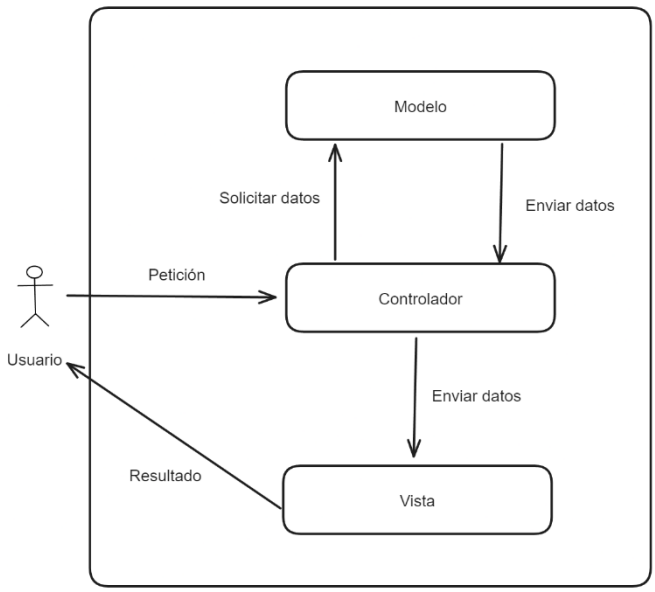


Figura 5 Modelo-Vista-Controlador

4.2 Implementación de la capa de seguridad

En esta sección, se describe la implementación de la capa de seguridad en la transferencia de datos a través de Bluetooth. Con el objetivo de garantizar la confidencialidad y la integridad de los archivos transmitidos, así como prevenir posibles vulnerabilidades en el proceso de transferencia de datos.

Para implementar la capa de seguridad en la transferencia de datos a través de Bluetooth, se ha utilizado la biblioteca Tink [38], que proporciona una solución robusta y fácil de usar para el cifrado y la autenticación de datos. Tink es una biblioteca de criptografía desarrollada por Google que ofrece un conjunto de APIs para diferentes algoritmos criptográficos y es recomendada para el desarrollo de aplicaciones seguras, en la Figura 6 Logo de Tink, se puede apreciar el logo o mascota de Tink.

En el contexto de la capa de seguridad implementada, Tink se utiliza para el cifrado simétrico de los datos transmitidos. Se ha utilizado el algoritmo de cifrado AesGcmJce [39], que ofrece una combinación eficiente de cifrado y autenticación de mensajes. AesGcmJce utiliza una clave compartida para cifrar y descifrar los datos, y proporciona integridad y confidencialidad en la transferencia de datos.

Figura 6 Logo de Tink

En el proceso de intercambio de claves, se ha implementado el protocolo de Diffie-Hellman [40] con curvas elípticas [41] (ECDH) para establecer un secreto compartido entre los dispositivos Bluetooth. La biblioteca java.security [42] se utiliza para generar las claves públicas y privadas del servidor y el cliente, mientras que la biblioteca javax.crypto [43] se utiliza para realizar las

operaciones de acuerdo de claves.

Al iniciar el servidor Bluetooth (startBluetoothServer), se realiza el intercambio seguro de claves públicas entre el servidor y el cliente. El servidor genera su par de claves utilizando la biblioteca KeyPairGenerator del paquete java.security, mientras que el cliente genera su par de claves de manera similar. Posteriormente, se realiza el intercambio de claves públicas a través del canal Bluetooth y se calcula el secreto compartido utilizando el algoritmo ECDH, en la Figura 7 Intercambio de claves servidor, se puede apreciar cómo se envían y reciben los datos necesarios para el intercambio de claves por parte del servidor. A continuación, se detalla paso a paso el código que establece la comunicación entre el cliente y el servidor Bluetooth, donde se intercambian las claves públicas del cliente y del servidor, y se genera el secreto compartido para la comunicación segura.

- bluetoothClientSocket?.outputStream?.apply { ... }: Este bloque de código utiliza la salida de datos del socket del cliente Bluetooth para enviar información al servidor.
- val clientPublicKeySizeBuffer = ByteArray(4): Se crea un búfer de tamaño 4 bytes para almacenar el tamaño de la clave pública del cliente.
- bluetoothClientSocket?.inputStream?.read(clientPublicKeySizeBuffer): Se lee el tamaño de la clave pública del cliente desde el flujo de entrada del socket del cliente.
- val clientPublicKeyBuffer = ByteArray(ByteBuffer.wrap(clientPublicKeySizeBuffer).int): Se crea un búfer para almacenar la clave pública del cliente utilizando el tamaño leído anteriormente.
- bluetoothClientSocket?.inputStream?.read(clientPublicKeyBuffer): Se lee la clave pública del cliente desde el flujo de entrada del socket del cliente y se almacena en el búfer.
- val keyFactory = KeyFactory.getInstance("EC"): Se obtiene una instancia de KeyFactory para el algoritmo de criptografía de curva elíptica (EC).
- val clientPublicKeySpec = X509EncodedKeySpec(clientPublicKeyBuffer): Se crea una instancia de X509EncodedKeySpec utilizando el búfer que contiene la clave pública del cliente.
- clientPublicKey = keyFactory.generatePublic(clientPublicKeySpec): Se genera la clave pública del cliente a partir de la especificación de la clave pública.
- val serverKeyAgreement: KeyAgreement = KeyAgreement.getInstance("ECDH"): Se obtiene una instancia de KeyAgreement para el algoritmo de acuerdo de claves de curva elíptica (ECDH) para el servidor.

- serverKeyAgreement.init(serverPrivateKey): Se inicializa el acuerdo de claves del servidor con la clave privada del servidor.
- serverKeyAgreement.doPhase(clientPublicKey, true): Se realiza la fase de acuerdo de claves del servidor con la clave pública del cliente.
- sharedSecretserver= serverKeyAgreement.generateSecret(): Se genera el secreto compartido del servidor.
- val serverPublicKeySizeBuffer = ByteBuffer.allocate(4).putInt(serverPublicKey.encoded.size).array(): Se crea un búfer que contiene el tamaño de la clave pública del servidor.
- bluetoothClientSocket?.outputStream?.write(serverPublicKeySizeBuffer): Se escribe el tamaño de la clave pública del servidor en el flujo de salida del socket del cliente.
- bluetoothClientSocket?.outputStream?.write(serverPublicKey.encoded): Se escribe la clave pública del servidor en el flujo de salida del socket del cliente.
- val serverSharedSecret128 = sharedSecretserver?.sliceArray(0 until 16): Se obtiene una porción de 16 bytes del secreto compartido del servidor.
- aead = AesGcmJce(serverSharedSecret128): Se crea una instancia de AesGcmJce para cifrar y descifrar datos utilizando el secreto compartido del servidor.

En el Apéndice 8.2 startBluetoothServer se puede apreciar todo el proceso desde que se inicia el servidor hasta que se descifran los datos para posteriormente guardar el archivo descifrado. En la Figura 8 Descifrando datos, se puede apreciar el código que a continuación se va a detallar, ya que fue uno de los principales problemas que se tuvo a la hora de implementar la capa de seguridad. Esta parte del código se encarga de recibir los datos enviados a través de Bluetooth, guardarlos en una lista de bytes y luego descifrarlos.

Aquí se explican los pasos en detalle:

- val allFileData: ArrayList<Byte> = ArrayList(): Se crea una lista allFileData para almacenar todos los datos recibidos como bytes.
- val fileBuffer = ByteArray(size = BUFFER_SIZE): Se crea un búfer de bytes con un tamaño definido por la constante BUFFER_SIZE. Este búfer se utiliza para leer los datos del flujo de entrada del socket Bluetooth.
- while (remainingBytes > 0) { ... }: Se inicia un bucle while que se ejecuta mientras queden bytes por leer (es decir, remainingBytes es mayor que 0).

- bytes = bluetoothClientSocket?.inputStream?.read(fileBuffer, 0, min(remainingBytes, BUFFER_SIZE)) ?: -1: Se lee un bloque de datos del flujo de entrada del socket Bluetooth y se almacena en el búfer fileBuffer. La cantidad de bytes leídos se guarda en la variable bytes.
- if (bytes == -1) { break }: Si no se han leído bytes (el valor de bytes es -1), se sale del bucle ya que no hay más datos para leer.
- remainingBytes -= bytes: Se resta la cantidad de bytes leídos (bytes) a la cantidad total de bytes restantes (remainingBytes).
- for (i in 0 until bytes) { allFileData.add(fileBuffer[i]) }: Se recorren los bytes leídos en el búfer fileBuffer y se añaden a la lista allFileData.
- val fileDataAsArray = allFileData.toByteArray(): Se convierte la lista de bytes allFileData en un array de bytes utilizando el método toByteArray().
- val plaintext = aead?.decrypt(fileDataAsArray, null): Se utiliza el objeto aead (instancia de AesGcmJce) para descifrar los datos recibidos. El método decrypt() se aplica al array de bytes fileDataAsArray.
- if (plaintext != null) { ... }: Si el descifrado es exitoso y se obtiene un texto plano (plaintext no es nulo), se llama a la función saveFile() para guardar el archivo con los datos descifrados.

A la hora de implementar esta parte del código no se tuvo en cuenta inicialmente que el método decrypt() de la instancia de AesGcmJce requiere recibir todos los datos cifrados a la vez para poder realizar el proceso de descifrado de manera adecuada. Una vez que se tienen todos los datos, se realiza el descifrado en un solo paso y se obtiene el texto plano. Si se intentara descifrar los datos a medida que se van leyendo, el resultado no sería correcto ya que el cifrado AES-GCM utiliza una combinación de cifrado y autenticación, y cada bloque de datos cifrados depende de los bloques anteriores para ser descifrados correctamente.

En la función connectToBluetoothDevice, se realiza un proceso similar de intercambio de claves públicas y cálculo del secreto compartido entre el cliente y el servidor. El cliente genera su par de claves y envía su clave pública al servidor a través del canal Bluetooth. El servidor utiliza la clave pública del cliente para calcular el secreto compartido utilizando ECDH. Este secreto compartido se utiliza posteriormente para la generación de la clave simétrica utilizada en el cifrado y descifrado de los datos, en la Figura 9 Intercambio de claves cliente, se puede apreciar cómo se envían y reciben los datos necesarios para el intercambio de claves por parte del cliente. A continuación, se detalla

paso a paso el código que se encarga de establecer la conexión con un dispositivo Bluetooth remoto y realizar un intercambio de claves públicas para establecer un secreto compartido, en el Apéndice 8.1 connectToBluetoothDevice, se puede apreciar el código completo de dicha función. Aquí se explica paso a paso:

- val clientPublicKeySizeBuffer = ByteBuffer.allocate(4).putInt(clientPublicKey.encoded?.size ?: 0).array(): Se crea un búfer de 4 bytes (clientPublicKeySizeBuffer) y se asigna el tamaño de la clave pública del cliente (clientPublicKey) en forma de array de bytes.
- val clientPublicKeyBuffer = clientPublicKey.encoded ?: ByteArray(0): Se obtiene la representación de bytes de la clave pública del cliente y se asigna al búfer clientPublicKeyBuffer. Si la clave pública es nula, se asigna un array de bytes vacío.
- socket.outputStream.write(clientPublicKeySizeBuffer): Se envía el tamaño de la clave pública del cliente al dispositivo remoto a través del flujo de salida del socket Bluetooth.
- socket.outputStream.write(clientPublicKeyBuffer): Se envía la clave pública del cliente al dispositivo remoto a través del flujo de salida del socket Bluetooth.
- val serverPublicKeySizeBuffer = ByteArray(4): Se crea un búfer de 4 bytes (serverPublicKeySizeBuffer) para recibir el tamaño de la clave pública del servidor desde el dispositivo remoto.
- socket.inputStream.read(serverPublicKeySizeBuffer): Se lee el tamaño de la clave pública del servidor desde el flujo de entrada del socket Bluetooth y se almacena en el búfer serverPublicKeySizeBuffer.
- val serverPublicKeySize = ByteBuffer.wrap(serverPublicKeySizeBuffer).int: Se obtiene el tamaño de la clave pública del servidor a partir de los bytes almacenados en serverPublicKeySizeBuffer.
- val serverPublicKeyBuffer = ByteArray(serverPublicKeySize): Se crea un búfer de bytes (serverPublicKeyBuffer) con el tamaño adecuado para recibir la clave pública del servidor desde el dispositivo remoto.
- socket.inputStream.read(serverPublicKeyBuffer): Se lee la clave pública del servidor desde el flujo de entrada del socket Bluetooth y se almacena en el búfer serverPublicKeyBuffer.
- val keyFactory = KeyFactory.getInstance("EC"): Se obtiene una instancia de la clase

KeyFactory para la generación de claves.

- val serverPublicKeySpec = X509EncodedKeySpec(serverPublicKeyBuffer): Se crea una especificación de clave pública a partir de los bytes de la clave pública del servidor.
- serverPublicKey = keyFactory.generatePublic(serverPublicKeySpec): Se genera la clave pública del servidor utilizando la especificación de clave pública y se asigna a la variable serverPublicKey.
- val clientKeyAgreement: KeyAgreement = KeyAgreement.getInstance("ECDH"): Se obtiene una instancia de la clase KeyAgreement para el algoritmo de acuerdo de clave ECDH (Diffie-Hellman elíptico).
- clientKeyAgreement.init(clientPrivateKey): Se inicializa el algoritmo de acuerdo de clave del cliente con la clave privada del cliente.
- clientKeyAgreement.doPhase(serverPublicKey, true): Se realiza la fase de acuerdo de clave con la clave pública del servidor y se especifica que se genera un secreto compartido.
- sharedSecretclient = clientKeyAgreement.generateSecret(): Se genera el secreto compartido a partir del acuerdo de claves realizado y se asigna a la variable sharedSecretclient.
- val clientSharedSecret128 = sharedSecretclient?.sliceArray(0 until 16): Se toman los primeros 16 bytes del secreto compartido y se asignan a clientSharedSecret128.
- aead = AesGcmJce(clientSharedSecret128): Se crea una instancia de AesGcmJce utilizando el secreto compartido como clave de cifrado y autenticación.

```
bluetoothClientSocket?.outputStream?.apply {
                        val clientPublicKeySizeBuffer = ByteArray(4)
                        bluetoothClientSocket?.inputStream?.read(clientPublicKeySizeBuffer)

                        val clientPublicKeyBuffer =
ByteArray(ByteBuffer.wrap(clientPublicKeySizeBuffer).int)
                        bluetoothClientSocket?.inputStream?.read(clientPublicKeyBuffer)

                        val keyFactory = KeyFactory.getInstance("EC")
                        val clientPublicKeySpec = X509EncodedKeySpec(clientPublicKeyBuffer)
                        clientPublicKey = keyFactory.generatePublic(clientPublicKeySpec)

                        val serverKeyAgreement: KeyAgreement = KeyAgreement.getInstance("ECDH")
                        serverKeyAgreement.init(serverPrivateKey)
                        serverKeyAgreement.doPhase(clientPublicKey, true)

                        sharedSecretserver= serverKeyAgreement.generateSecret()

                        val serverPublicKeySizeBuffer =
ByteBuffer.allocate(4).putInt(serverPublicKey.encoded.size).array()
                        bluetoothClientSocket?.outputStream?.write(serverPublicKeySizeBuffer)

                        bluetoothClientSocket?.outputStream?.write(serverPublicKey.encoded)

                        val serverSharedSecret128 = sharedSecretserver?.sliceArray(0 until 16)

                        aead = AesGcmJce(serverSharedSecret128)


                    }
```

Figura 7 Intercambio de claves servidor

```
                            val allFileData: ArrayList<Byte> = ArrayList()
                            val fileBuffer = ByteArray(size = BUFFER_SIZE)
                            while (remainingBytes > 0){
                                bytes =
bluetoothClientSocket?.inputStream?.read(fileBuffer,0,min(remainingBytes,BUFFER_SIZE)) ?: -1

                                if (bytes == -1) {
                                    break
                                }
                                remainingBytes -= bytes
                                // Guardar los datos recibidos
                                for (i in 0 until bytes) {
                                    allFileData.add(fileBuffer[i])
                                }
                            }
                            // Cuando ya no queda nada por leer, es momento de descifrar los datos.
                            val fileDataAsArray = allFileData.toByteArray()
                            val plaintext = aead?.decrypt(fileDataAsArray, null)

                            if (plaintext != null) {
                                saveFile(context, plaintext, fileName)
                            } else {
                                Log.e("Bluetooth", "Error al desencriptar los datos en
startBluetoothServer")
                            }
```

Figura 8 Descifrando datos

```
val clientPublicKeySizeBuffer = ByteBuffer.allocate(4).putInt(clientPublicKey.encoded?.size ?: 0).array()

                val clientPublicKeyBuffer = clientPublicKey.encoded ?: ByteArray(0)

                socket.outputStream.write(clientPublicKeySizeBuffer)

                socket.outputStream.write(clientPublicKeyBuffer)


                val serverPublicKeySizeBuffer = ByteArray(4)
                socket.inputStream.read(serverPublicKeySizeBuffer)

                val serverPublicKeySize = ByteBuffer.wrap(serverPublicKeySizeBuffer).int
                val serverPublicKeyBuffer = ByteArray(serverPublicKeySize)
                socket.inputStream.read(serverPublicKeyBuffer)

                val keyFactory = KeyFactory.getInstance("EC")
                val serverPublicKeySpec = X509EncodedKeySpec(serverPublicKeyBuffer)
                serverPublicKey = keyFactory.generatePublic(serverPublicKeySpec)

                val clientKeyAgreement: KeyAgreement = KeyAgreement.getInstance("ECDH")
                clientKeyAgreement.init(clientPrivateKey)
                clientKeyAgreement.doPhase(serverPublicKey, true)

                sharedSecretclient = clientKeyAgreement.generateSecret()
                val clientSharedSecret128 = sharedSecretclient?.sliceArray(0 until 16)

                aead = AesGcmJce(clientSharedSecret128)
```

Figura 9 Intercambio de claves cliente

4.3 Interfaz de usuario y diseño

En esta sección, se aborda el proceso de diseño de la interfaz de usuario de la aplicación, así como la consideración de principios de accesibilidad y la elección de una paleta de colores adecuada.

Para comenzar, se realizaron mockups en la herramienta Excalidraw [35] de las posibles vistas iniciales que serían necesarias en la aplicación, en la Figura 10 Mockups, se pueden apreciar. Estos mockups proporcionaron una representación visual de cómo se estructuraría la interfaz y permitieron explorar diferentes opciones de diseño básicas. Con base en estos mockups, se procedió a crear diseños más detallados utilizando la herramienta Figma [32]. Esto brindó una visión más clara de cómo se vería la interfaz final y permitió evaluar la coherencia visual y la usabilidad de los elementos.

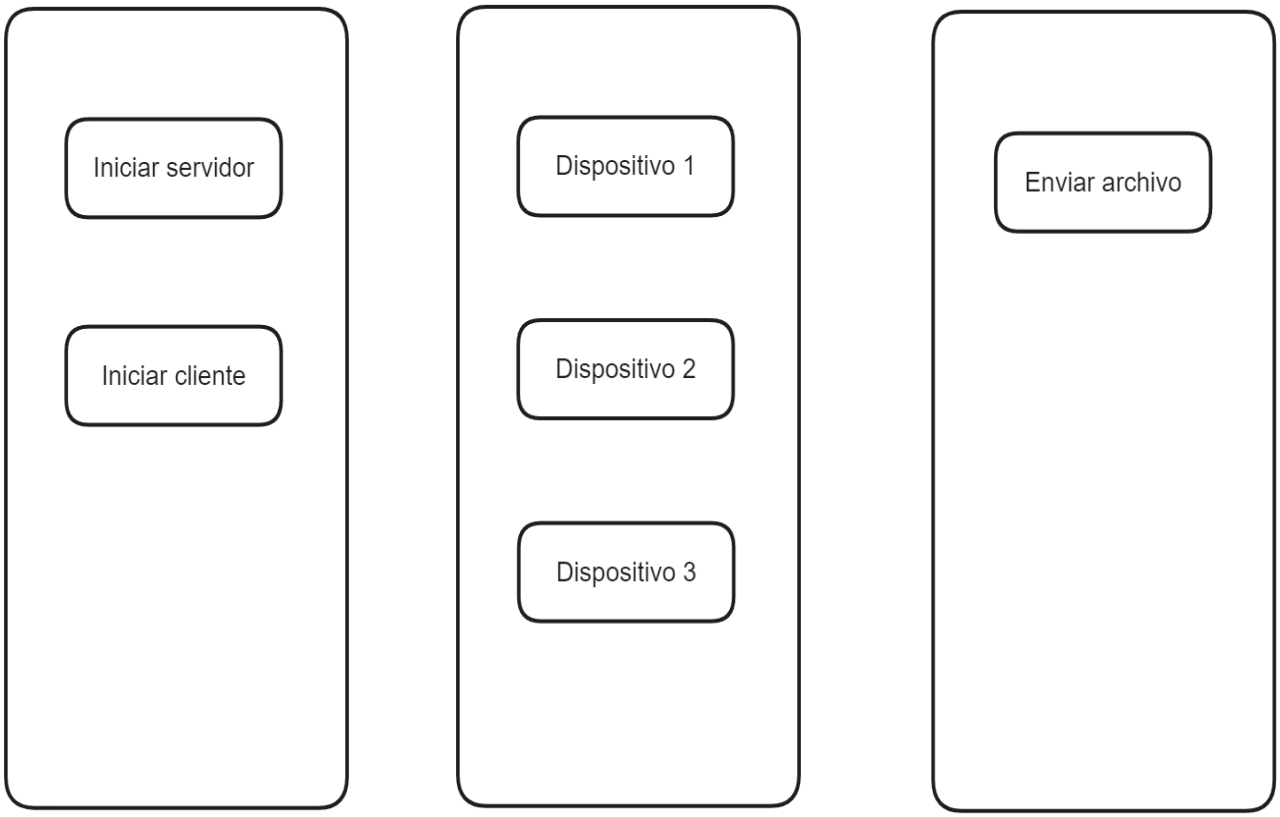


Figura 10 Mockups

Además del aspecto visual, se dedicó especial atención a la accesibilidad de la aplicación. Se investigaron conceptos y pautas de Accesibilidad Móvil proporcionados por el W3C (World Wide Web Consortium) [44]. Estos recursos ayudaron a comprender cómo garantizar que la aplicación fuera inclusiva y pudiera ser utilizada por personas con diferentes capacidades. Se prestó especial atención a criterios de contraste, legibilidad de texto, tamaños de fuente adaptables y manejo adecuado de la navegación y el enfoque.

En cuanto a la elección de colores, se optó por una gama que cumpliera con los criterios de contraste establecidos en las pautas de accesibilidad. Esto garantiza que los elementos visuales, como el texto y los botones, sean legibles y distinguibles para todos los usuarios, independientemente de sus capacidades visuales. Se consideraron cuidadosamente las combinaciones de colores y se realizaron pruebas de contraste para asegurar un diseño accesible y atractivo.

Para validar la paleta de colores y asegurarse de que se cumplieran los criterios de accesibilidad, se utilizó la aplicación de Google llamada "Test de Accesibilidad"[35].

Una vez establecida la base de diseño, se procedió a implementar la interfaz de usuario utilizando Jetpack Compose, que gracias a que ofrece una amplia gama de componentes y elementos visuales predefinidos por su compatibilidad con Material Design [45], que es un conjunto de directrices de diseño desarrollado por Google que ofrece una experiencia visual coherente y atractiva en las aplicaciones de Android, logré acelerar el proceso de desarrollo y mejorar la consistencia visual de la aplicación. También proporciona una actualización en tiempo real de la interfaz, lo que me permite una vista previa prácticamente instantánea de los cambios realizados en el código.

4.4 Actividad principal

Aquí detallaré cómo se desarrolló la MainActivity, que es el punto de entrada principal de la aplicación y se encarga de administrar el ciclo de vida de la actividad y la interacción con el usuario.

Para comenzar, se realizó una investigación sobre la navegación en Android utilizando Jetpack Compose. Se descubrió que Jetpack Compose proporciona NavController [46], que es una herramienta para configurar la navegación en la aplicación. Se decidió utilizar NavController para definir las rutas y la navegación entre las diferentes pantallas de la aplicación. Se creó una ruta para cada pantalla que se desarrolló, lo que permitió una transición fluida entre ellas.

Además de la navegación, se consideró la situación en la que el usuario no tenga el Bluetooth activado en su dispositivo. Se reconoció que el Bluetooth es necesario para el correcto funcionamiento de la aplicación, por lo tanto, se implementó un código específico para comprobar si el Bluetooth está activado. Si se detecta que el Bluetooth está desactivado, se muestra una solicitud al usuario para que lo active. Esto se logró mediante el uso de ActivityResultContracts.StartActivityForResult [27], que es una API de Jetpack [47] que permite solicitar actividades y recibir resultados.

Además, se desarrolló un código específico para detectar si el Bluetooth se apaga en algún momento mientras la aplicación está en ejecución. Esto se logró registrando un receptor de difusión (BroadcastReceiver [48]) para las acciones de cambio de estado del Bluetooth. Si se detecta que el Bluetooth se apaga, se vuelve a solicitar al usuario que lo active.

4.5 Desarrollo de la capa de presentación

En esta sección, se abordará el proceso de desarrollo de la capa de presentación de la aplicación, centrándose en la implementación de las diferentes vistas (composables) que componen la interfaz de usuario. Cada vista representa un componente visual específico, como un botón, un cuadro de texto o una lista, y define su apariencia y comportamiento. Las figuras que se muestran en esta sección pueden tener cambios respecto al diseño final.

A continuación, se proporcionará una descripción detallada de cada vista implementada en la aplicación, destacando su propósito y funcionalidad en el flujo de interacción. Además, se explicará cómo se utilizó Jetpack Compose para definir y gestionar estas vistas de manera declarativa, aprovechando las ventajas que ofrece esta biblioteca de interfaz de usuario.

CustomAppBar:

El composable CustomAppBar crea una barra superior personalizada para la aplicación. Puede mostrar un título y un icono de navegación en la parte izquierda de la barra superior. Si se proporciona un icono de navegación y una acción de navegación, el icono será interactivo y ejecutará la acción al hacer clic en él. Si no se proporciona un icono de navegación, la barra superior solo mostrará el título.

MainScreen:

El composable MainScreen representa la pantalla principal de la aplicación y muestra dos botones interactivos que permiten al usuario seleccionar entre enviar archivos y recibir archivos. Los botones están estilizados con colores y tamaños personalizados para una apariencia atractiva y utilizan el NavHostController para la navegación entre las diferentes pantallas de la aplicación, en la Figura 11 Pantalla principal, se puede ver una aproximación al diseño final.

Figura 11 Pantalla principal

BluetoothDeviceListScreen:

El composable BluetoothDeviceListScreen muestra una lista de dispositivos Bluetooth emparejados y permite al usuario seleccionar uno para establecer una conexión. Una vez que se establece la

conexión exitosamente, se realiza la navegación a la pantalla cliente, en la Figura 12 Listado de dispositivos emparejados, se puede apreciar un diseño aproximado.

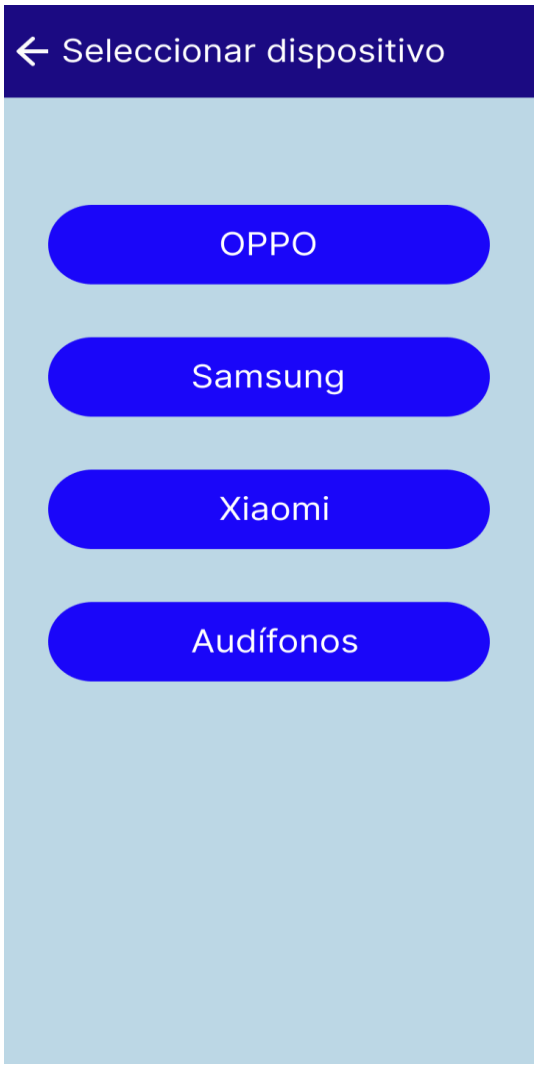


Figura 12 Listado de dispositivos emparejados

ClientScreen:

El composable ClientScreen muestra la pantalla del cliente para seleccionar y enviar un archivo mediante Bluetooth. Permite al usuario seleccionar un archivo y muestra el nombre del archivo seleccionado. Al hacer clic en el botón "Enviar archivo", se envía el archivo al dispositivo Bluetooth conectado, en la Figura 13 Pantallas para enviar datos, se puede apreciar un diseño aproximado.

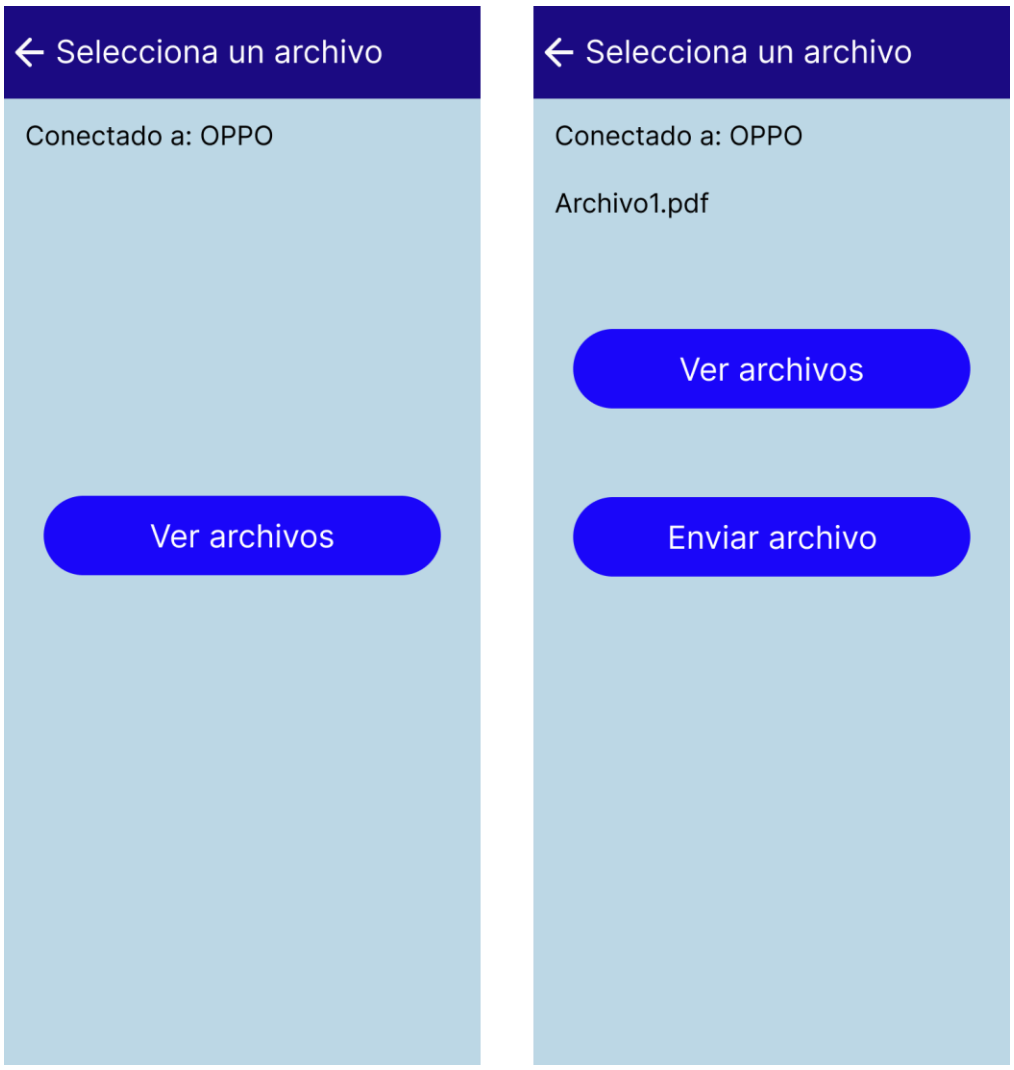


Figura 13 Pantallas para enviar datos

ServerScreen:

El composable ServerScreen muestra la pantalla del servidor para iniciar el servidor Bluetooth y cerrar la conexión Bluetooth. Permite al usuario administrar la conexión Bluetooth, en la Figura 14 Pantalla para recibir datos, se puede apreciar un diseño aproximado.

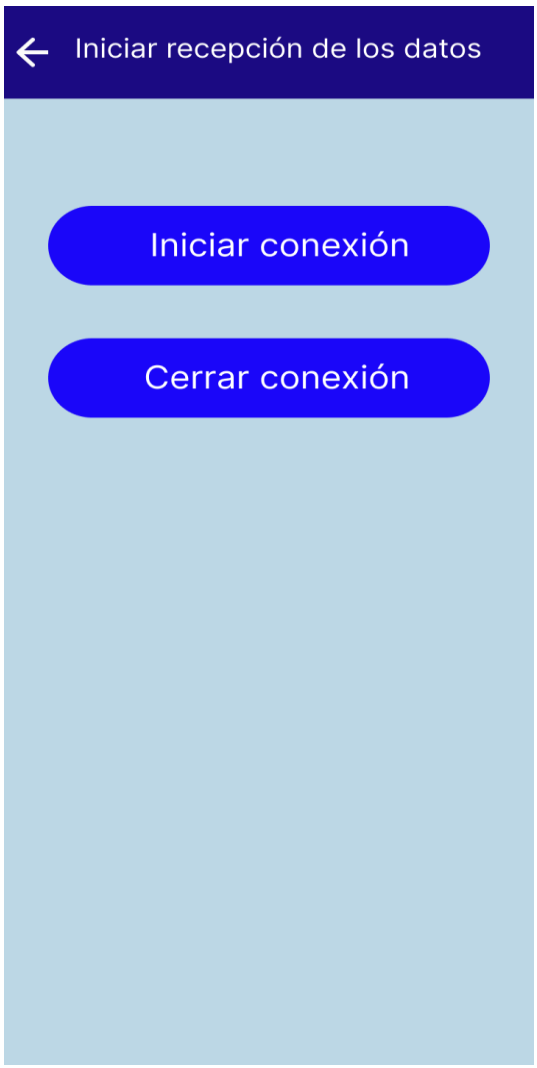


Figura 14 Pantalla para recibir datos

4.6 Desarrollo del Controlador Bluetooth

En el contexto del desarrollo de software, un controlador es un componente o módulo que se encarga de interactuar con dispositivos o sistemas externos, gestionando su funcionalidad y proporcionando una interfaz para que otros componentes de la aplicación puedan utilizarlo de manera simplificada.

En el caso específico del Controlador Bluetooth de esta aplicación, su objetivo es facilitar la comunicación y transferencia de datos a través de Bluetooth. Se encarga de administrar la conectividad Bluetooth, permitiendo la conexión con dispositivos emparejados, establecer un servidor Bluetooth para recibir archivos y enviar datos a través de Bluetooth.

La API de Bluetooth proporcionada por Android es utilizada para interactuar con las

capacidades de Bluetooth del dispositivo. El controlador hace uso de esta API para realizar tareas como obtener los dispositivos emparejados, iniciar un servidor Bluetooth y establecer una conexión con un dispositivo Bluetooth remoto. A continuación, se describen diferentes partes del código:

- Inicialización y actualización de dispositivos emparejados: El controlador inicia con la inicialización del adaptador Bluetooth y el gestor de servicios Bluetooth. A través de updatePairedDevices(), se actualiza la lista de dispositivos Bluetooth emparejados, obteniendo el nombre y la dirección de cada dispositivo emparejado.
- Inicio del servidor Bluetooth: El método startBluetoothServer() inicia el servidor Bluetooth mediante la creación de un socket del servidor Bluetooth. El servidor se pone en modo de escucha para aceptar conexiones entrantes de clientes Bluetooth. Una vez que se establece la conexión con un cliente, se cierra el socket del servidor y se inicia un hilo para escuchar los mensajes entrantes del cliente.
- Conexión con un dispositivo Bluetooth remoto: El método connectToBluetoothDevice() establece una conexión con un dispositivo Bluetooth remoto específico. Se crea un socket Bluetooth y se intenta la conexión. Si la conexión se establece correctamente, se notifica al listener de conexión exitosa. En caso de error, se cierra el socket y se notifica al listener de error de conexión.
- Cierre de la conexión: El método closeConnection() se utiliza para cerrar la conexión tanto del servidor Bluetooth como del cliente Bluetooth. Se cierra el socket del servidor y se establece el socket del cliente a nulo.
- Configuración de la escucha de datos: El método setDataListener() permite configurar un listener de datos Bluetooth. Este listener se utiliza para recibir y manejar los datos enviados a través de Bluetooth.
- Envío de datos a través de Bluetooth: El método sendData() se utiliza para enviar datos a través de Bluetooth desde el cliente al servidor. Los datos se envían mediante el uso de un OutputStream del socket Bluetooth. Antes de enviar los datos, se muestra el nombre del archivo que se está enviando mediante el listener de datos. Los datos se envían en varias partes para asegurar una transferencia completa del archivo.
- Guardar archivo recibido: El método saveFile() se encarga de construir y guardar el archivo recibido en el dispositivo. Se utiliza el directorio de almacenamiento externo de la aplicación para crear un directorio y escribir los datos recibidos en un archivo específico.

Capítulo 5 Conclusiones y líneas futuras

En este capítulo, se presentan las conclusiones obtenidas a partir del desarrollo del proyecto y se proponen líneas futuras de trabajo para mejorar y ampliar la funcionalidad de la aplicación BlueShare.

La realización de este proyecto permitió adquirir conocimientos y habilidades en diversas tecnologías y conceptos, como la programación en Android, el uso de la tecnología Bluetooth y la implementación de primitivas criptográficas para garantizar la seguridad en la transferencia de datos.

El análisis del estado del arte en cuanto a transferencia segura de datos y tecnología Bluetooth proporcionó una base sólida para el diseño e implementación de la aplicación. El estudio detallado de la tecnología Bluetooth permitió comprender sus características, limitaciones y protocolos asociados. Esto fue fundamental para diseñar una solución eficiente y segura de transferencia de datos.

Se logró desarrollar una aplicación funcional que permite a los usuarios transferir archivos entre dispositivos mediante Bluetooth, incorporando una capa adicional de seguridad mediante el cifrado de los datos enviados y su posterior descifrado en el dispositivo receptor. Pero también durante el desarrollo del proyecto, se enfrentaron desafíos significativos, como el desconocimiento inicial de las tecnologías utilizadas y la resolución de problemas relacionados con el cifrado de los datos. Sin embargo, a través del análisis minucioso, la dedicación y ayuda de los tutores, se superaron estos obstáculos y se logró implementar con éxito la capa de seguridad.

Si bien se lograron los objetivos principales del proyecto, existen diversas oportunidades de mejora y líneas futuras de trabajo que podrían explorarse. A continuación, se proponen algunas de estas líneas futuras:

- Mejora en el diseño y la usabilidad: La aplicación podría beneficiarse de mejoras en el diseño visual y la experiencia de usuario. Se podría realizar un análisis de usabilidad y realizar ajustes en la interfaz para hacerla más intuitiva y fácil de usar.
- Refactorización y modularización del código: Para mejorar la mantenibilidad y escalabilidad del código, se recomienda realizar una refactorización y modularización del mismo. Esto permitiría una mejor organización y estructura del código, facilitando su comprensión y futuras modificaciones.
- Inyección de dependencias: Se investigó el uso de Hilt [49], pero no dio tiempo de

implementarlo. La implementación de la inyección de dependencias permitiría reducir la dependencia directa entre las clases y mejorar la testabilidad del código. Esto facilitaría la incorporación de pruebas unitarias y simplificaría la introducción de nuevas funcionalidades en el futuro.

- Incorporación de pruebas unitarias: Se deberían crear pruebas unitarias para verificar el correcto funcionamiento de los componentes clave de la aplicación. Esto garantizaría la calidad y estabilidad del código, así como facilitaría la detección temprana de posibles errores o fallos.
- Selección de primitivas criptográficas: Actualmente, la aplicación utiliza una primitiva criptográfica específica para el cifrado de los datos. Sería interesante permitir al usuario seleccionar la primitiva criptográfica que desee utilizar, brindando opciones adicionales de seguridad y adaptabilidad a diferentes escenarios, en la Figura 15 Caso de uso con selección de primitiva, se puede apreciar el diagrama de caso de uso de la transferencia de datos permitiéndole al cliente la selección de la primitiva a utilizar.
- Integración de funciones adicionales de seguridad: Se podrían integrar funciones adicionales de seguridad, como la autenticación mutua entre dispositivos, el establecimiento de claves de sesión temporales o la detección de ataques o intrusiones en la comunicación.

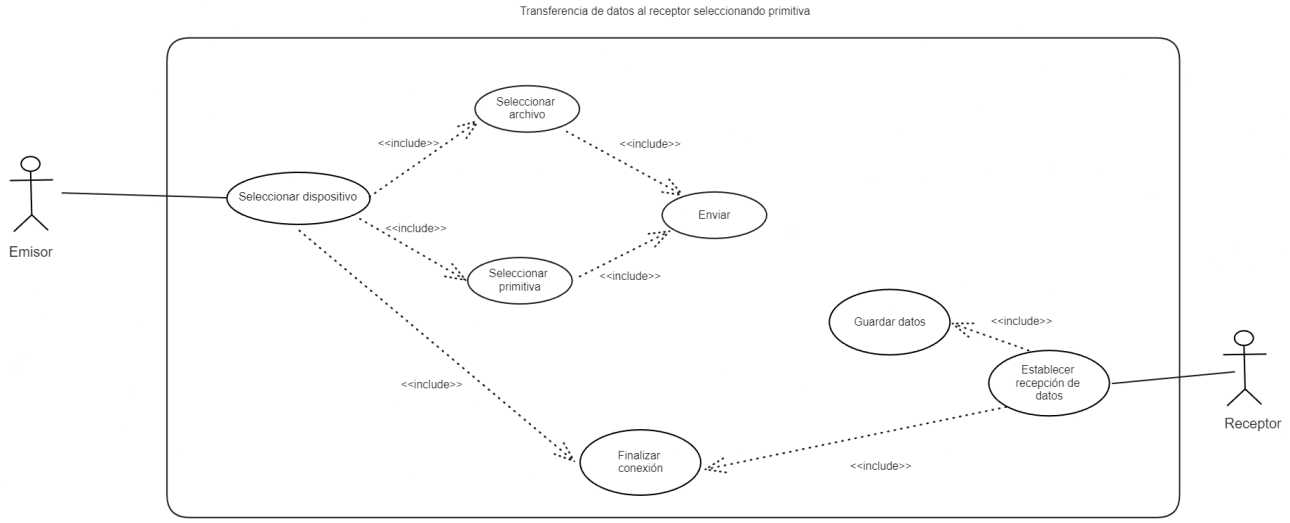


Figura 15 Caso de uso con selección de primitiva

Capítulo 6 Summary and Conclusions

In this chapter, we present the conclusions drawn from the project development and propose future lines of work to enhance and expand the functionality of the BlueShare application.

The completion of this project allowed for the acquisition of knowledge and skills in various technologies and concepts, such as Android programming, the use of Bluetooth technology, and the implementation of cryptographic primitives to ensure data security.

The analysis of the state-of-the-art in secure data transfer and Bluetooth technology provided a solid foundation for the design and implementation of the application. A detailed study of Bluetooth technology enabled a comprehensive understanding of its characteristics, limitations, and associated protocols. This understanding was crucial in designing an efficient and secure solution for data transfer.

A functional application was successfully developed, enabling users to transfer files between devices via Bluetooth while incorporating an additional layer of security through data encryption and decryption on the receiving device. However, significant challenges were encountered during the project, including the initial unfamiliarity with the technologies used and the resolution of issues related to data encryption. Through meticulous analysis, dedication, and guidance from mentors, these obstacles were overcome, and the security layer was successfully implemented.

While the main objectives of the project were achieved, several opportunities for improvement and future lines of work can be explored. The following are some proposed future directions:

- Improvement in design and usability: The application could benefit from enhancements in visual design and user experience. Conducting a usability analysis and making adjustments to the interface to make it more intuitive and user-friendly would be valuable.
- Code refactoring and modularization: To enhance code maintainability and scalability, it is recommended to refactor and modularize the codebase. This would lead to better code organization and structure, facilitating comprehension and future modifications.
- Dependency injection: Investigation into the use of Hilt [51] was initiated but could not be implemented within the given timeframe. Implementing dependency injection would reduce direct dependency between classes and improve code testability. This would facilitate the incorporation of unit tests and simplify the introduction of new features in the

future.

- Incorporation of unit testing: Creating unit tests to verify the proper functioning of key components in the application is essential. This would ensure code quality and stability, as well as facilitate the early detection of potential errors or failures.
- Selection of cryptographic primitives: Currently, the application utilizes a specific cryptographic primitive for data encryption. Allowing users to select the cryptographic primitive of their choice would provide additional security options and adaptability to different scenarios.
- Integration of additional security features: Additional security features could be integrated, such as mutual device authentication, temporary session key establishment, or detection of communication attacks or intrusions.

Capítulo 7 Presupuesto

Se detallarán los aspectos financieros relacionados con el desarrollo del proyecto. Aquí se presentan los costos estimados asociados a diferentes elementos, como coste personal, hardware y software.

7.1 Coste personal

Teniendo en cuenta que un ingeniero informático cobra alrededor de 15€ por hora trabajada, en la Tabla 7.1 Coste personal, se ve el coste personal que se estima para el desarrollo de la aplicación en función a la cantidad de horas dedicada a una tarea.

Tareas	Horas	Coste (€)
Análisis de la tecnología Bluetooth	65	975
Análisis de requisitos	20	300
Planificación del proyecto	35	525
Diseño de la interfaz	15	225
Pruebas y depuración	25	375
Implementación	200	3000
Redacción de la memoria	72	1080
Total	432	6480

Tabla 7.1Coste personal

7.2 Coste de hardware

En la Tabla 7.2 Coste de hardware se puede apreciar los costes de los componentes hardware utilizados durante el desarrollo del trabajo.

Componente	Coste (€)
Smartphone OPPO A73 5G	300
Smartphone Xiaomi redmi note 9 PRO	250
Ordenador de sobremesa	1000
Total	1550

Tabla 7.2 Coste de hardware

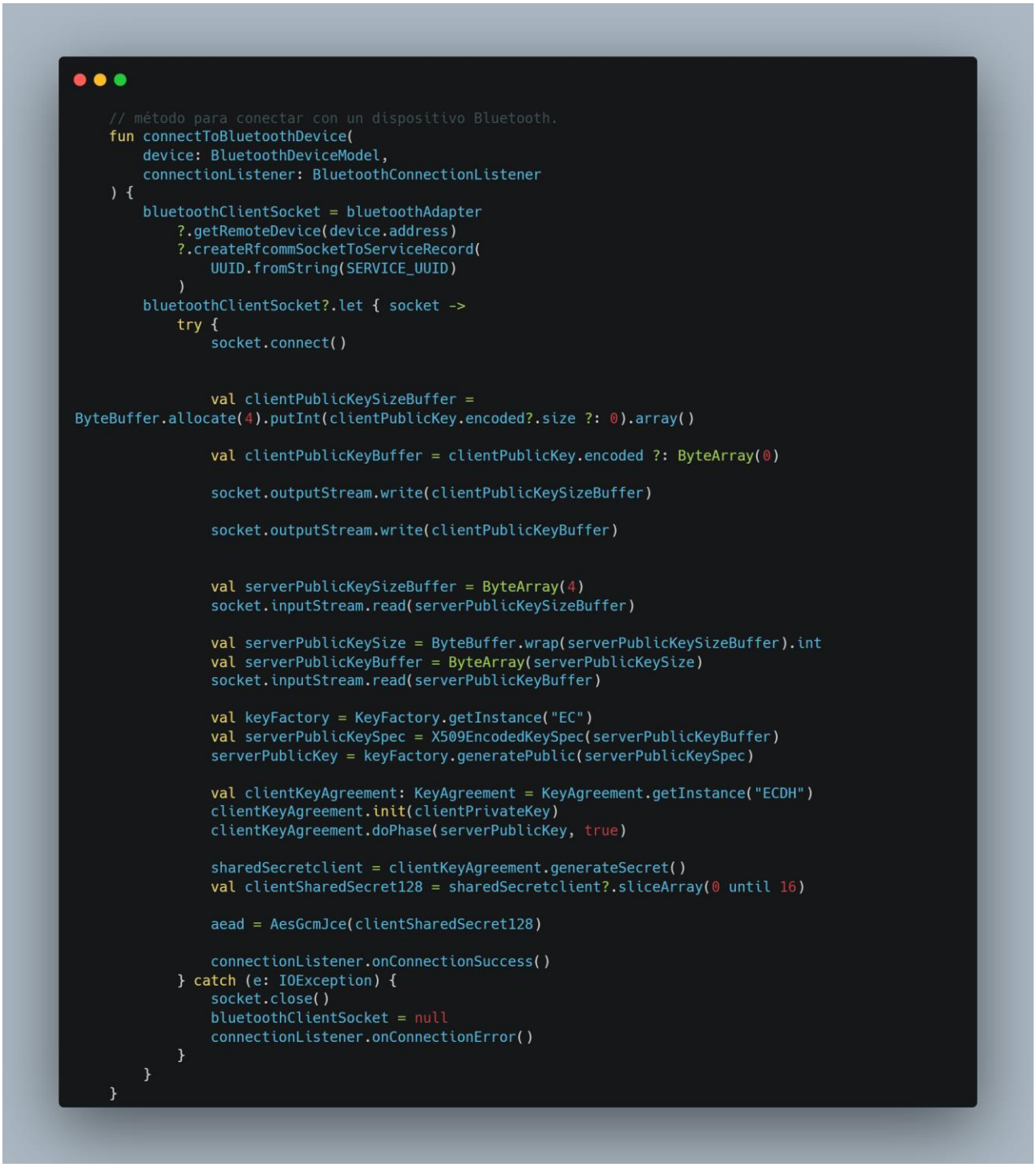
7.3 Coste Software

En este caso el coste de software a sido 0, ya que todas las herramientas usadas son gratuitas o al menos el uso que se hizo de ellas no requirieron un pago económico, en el caso del Sistema Operativo se uso Windows 10, pero todas las herramientas utilizadas se pueden usar en Linux para evitar el pago por la licencia de Windows.

Capítulo 8 Apéndice 1

8.1 connectToBluetoothDevice

```
/**********************************************************************************

* BluetoothController.kt

**********************************************************************************

* Milton Daniel Rivas Quintero

* 14/07/2023

* Este código establece una conexión Bluetooth con un dispositivo remoto, realiza un intercambio de claves

* públicas y genera un secreto compartido para establecer una comunicación segura. También configura la

* capa de seguridad para cifrar y descifrar los datos en la conexión Bluetooth.

**********************************************************************************/
```



```
    // método para conectar con un dispositivo Bluetooth.
    fun connectToBluetoothDevice(
        device: BluetoothDeviceModel,
        connectionListener: BluetoothConnectionListener
    ) {
        bluetoothClientSocket = bluetoothAdapter
            ?.getRemoteDevice(device.address)
            ?.createRfcommSocketToServiceRecord(
                UUID.fromString(SERVICE_UUID)
            )
        bluetoothClientSocket?.let { socket ->
            try {
                socket.connect()


                val clientPublicKeySizeBuffer =
ByteBuffer.allocate(4).putInt(clientPublicKey.encoded?.size ?: 0).array()

                val clientPublicKeyBuffer = clientPublicKey.encoded ?: ByteArray(0)

                socket.outputStream.write(clientPublicKeySizeBuffer)

                socket.outputStream.write(clientPublicKeyBuffer)


                val serverPublicKeySizeBuffer = ByteArray(4)
                socket.inputStream.read(serverPublicKeySizeBuffer)

                val serverPublicKeySize = ByteBuffer.wrap(serverPublicKeySizeBuffer).int
                val serverPublicKeyBuffer = ByteArray(serverPublicKeySize)
                socket.inputStream.read(serverPublicKeyBuffer)

                val keyFactory = KeyFactory.getInstance("EC")
                val serverPublicKeySpec = X509EncodedKeySpec(serverPublicKeyBuffer)
                serverPublicKey = keyFactory.generatePublic(serverPublicKeySpec)

                val clientKeyAgreement: KeyAgreement = KeyAgreement.getInstance("ECDH")
                clientKeyAgreement.init(clientPrivateKey)
                clientKeyAgreement.doPhase(serverPublicKey, true)

                sharedSecretclient = clientKeyAgreement.generateSecret()
                val clientSharedSecret128 = sharedSecretclient?.sliceArray(0 until 16)

                aead = AesGcmJce(clientSharedSecret128)

                connectionListener.onConnectionSuccess()
            } catch (e: IOException) {
                socket.close()
                bluetoothClientSocket = null
                connectionListener.onConnectionError()
            }
        }
    }
```

Figura 16 connectToBluetoothDevice

8.2 startBluetoothServer

```
/**********************************************************************************

 * BluetoothController.kt

 **********************************************************************************

 * Milton Daniel Rivas Quintero

 * 14/07/2023

 * Este código establece y administra la conexión Bluetooth como un servidor, realiza el intercambio de claves

 * públicas y secreto compartido, y recibe y descifra los datos entrantes del cliente Bluetooth.

 **********************************************************************************/
```

```
    // método para iniciar el servidor Bluetooth.
    fun startBluetoothServer() {
        bluetoothServerSocket = bluetoothAdapter?.listenUsingRfcommWithServiceRecord(
            "BluetoothServer",
            UUID.fromString(SERVICE_UUID)
        )

        var shouldLoop = true
        while(shouldLoop){
            try {
                bluetoothServerSocket?.accept()?.let { socket ->

                    bluetoothServerSocket?.close()
                    bluetoothServerSocket = null
                    bluetoothClientSocket = socket
                    bluetoothClientSocket?.outputStream?.apply {
                        val clientPublicKeySizeBuffer = ByteArray(4)
                        bluetoothClientSocket?.inputStream?.read(clientPublicKeySizeBuffer)

                        val clientPublicKeyBuffer =
ByteArray(ByteBuffer.wrap(clientPublicKeySizeBuffer).int)
                        bluetoothClientSocket?.inputStream?.read(clientPublicKeyBuffer)

                        val keyFactory = KeyFactory.getInstance("EC")
                        val clientPublicKeySpec = X509EncodedKeySpec(clientPublicKeyBuffer)
                        clientPublicKey = keyFactory.generatePublic(clientPublicKeySpec)

                        val serverKeyAgreement: KeyAgreement = KeyAgreement.getInstance("ECDH")
                        serverKeyAgreement.init(serverPrivateKey)
                        serverKeyAgreement.doPhase(clientPublicKey, true)

                        sharedSecretserver= serverKeyAgreement.generateSecret()

                        val serverPublicKeySizeBuffer =
ByteBuffer.allocate(4).putInt(serverPublicKey.encoded.size).array()
                        bluetoothClientSocket?.outputStream?.write(serverPublicKeySizeBuffer)

                        bluetoothClientSocket?.outputStream?.write(serverPublicKey.encoded)

                        val serverSharedSecret128 = sharedSecretserver?.sliceArray(0 until 16)

                        aead = AesGcmJce(serverSharedSecret128)


                    }
                    shouldLoop = false
                    Toast.makeText(context, "Conexión con cliente establecida",
Toast.LENGTH_SHORT).show()
                }

                // Hilo para escuchar los mensajes entrantes del cliente
                Thread {
                    while (true) {
                        try {
                            // Leer el tamaño nombre del archivo
                            val fileNameSizeBuffer = ByteArray(4)
                            bluetoothClientSocket?.inputStream?.read(fileNameSizeBuffer)
                            val fileNameSize = ByteBuffer.wrap(fileNameSizeBuffer).int
                            // Leer el nombre del archivo
                            val fileNameBuffer = ByteArray(fileNameSize)
                            bluetoothClientSocket?.inputStream?.read(fileNameBuffer)
                            val fileName = String(fileNameBuffer)

                            // Leer el tamaño del archivo
                            val fileSizeBuffer = ByteArray(4)
                            bluetoothClientSocket?.inputStream?.read(fileSizeBuffer)
                            val fileSize = ByteBuffer.wrap(fileSizeBuffer).int
                            var bytes: Int
                            var remainingBytes = fileSize
                            val allFileData: ArrayList<Byte> = ArrayList()
                            val fileBuffer = ByteArray(size = BUFFER_SIZE)
                            while (remainingBytes > 0){
                                bytes =
bluetoothClientSocket?.inputStream?.read(fileBuffer,0,min(remainingBytes,BUFFER_SIZE)) ?: -1

                                if (bytes == -1) {
                                    break
                                }
                                remainingBytes -= bytes
                                // Guardar los datos recibidos
                                for (i in 0 until bytes) {
                                    allFileData.add(fileBuffer[i])
                                }
                            }
                            // Cuando ya no queda nada por leer, es momento de descifrar los datos.
                            val fileDataAsArray = allFileData.toByteArray()
                            val plaintext = aead?.decrypt(fileDataAsArray, null)

                            if (plaintext != null) {
                                saveFile(context, plaintext, fileName)
                            } else {
                                Log.e("Bluetooth", "Error al desencriptar los datos en
startBluetoothServer")
                            }
                        } catch (e: IOException) {
                            // Error al leer los datos
                            Log.e("Bluetooth", "Error al leer los datos en startBluetoothServer", e)
                            break
                        }
                    }
                }.start()
            }catch (e: IOException){
                shouldLoop = false
            }
        }
    }
```

Figura 17 startBluetoothServer

Capítulo 9 Bibliografía

[1]. David Sandoval, Jesús María Gómez. Introducción a los ataques Bluetooth. 2021. url: https://www.tarlogic.com/es/blog/introduccion-a-los-ataques-bluetooth/

[2]. INCIBE-CERT. Múltiples vulnerabilidades en dispositivos que soportan especificaciones Bluetooth. 2021. url: https://www.incibe-cert.es/alerta-temprana/avisos-seguridad/multiples-vulnerabilidades-dispositivos-soportan-especificaciones

[3]. CERT. Devices supporting Bluetooth Core and Mesh Specifications are vulnerable to impersonation attacks and AuthValue disclosure. 2021. url: https://kb.cert.org/vuls/id/799380

[4]. Bluetooth. Core Specification 5.2. 202. url: https://www.bluetooth.com/specifications/specs/core-specification-5-2/

[5]. Bluetooth. Mesh Profile 1.0.1. url: https://www.bluetooth.com/specifications/specs/mesh-profile-1-0-1/

[6]. La Sexta. Amazon resuelve una vulnerabilidad de los Echo que evita que se "auto hackeen". 2022. url: https://www.lasexta.com/tecnologia-tecnoxplora/gadgets/amazon-resuelve-vulnerabilidad-echo-que-evita-que-auto-hackeen_20220307622605a55bbac90001713b7d.html

[7]. arXiv. Alexa versus Alexa: Controlling Smart Speakers by Self-Issuing Voice Commands. 2022. url: https://arxiv.org/pdf/2202.08619.pdf

[8]. Orange. BrakTooth, la vulnerabilidad de 'bluetooth' que puede bloquear millones de dispositivos. 2021. url: https://blog.orange.es/consejos-y-trucos/braktooth/

[9]. espressif. ESP32. url: https://www.espressif.com/en/products/socs/esp32

[10]. BrakTooth: Causing Havoc on Bluetooth Link Manager. url: https://dl.packetstormsecurity.net/papers/general/braktooth.pdf

[11]. Bluetooth. Reporting Security Vulnerabilities url: https://www.bluetooth.com/learn-about-bluetooth/key-attributes/bluetooth-security/reporting-security/

[12]. CVE Details. CVE-2021-28139. 2021. url: https://www.cvedetails.com/cve/CVE-2021-28139/

[13]. CVE Details. CVE-2020-26558. 2022. url: https://www.cvedetails.com/cve/CVE-2020-26558/

[14]. CVE Details. CVE-2020-26559. 2021. url: https://www.cvedetails.com/cve/CVE-

2020-26559/

[15]. CVE Details. CVE-2020-26560. 2021. url: https://www.cvedetails.com/cve/CVE-2020-26560/

[16]. CVE Details. CVE-2020-26555. 2022. url: https://www.cvedetails.com/cve/CVE-2020-26555/

[17]. CVE Details. CVE-2020-10135. 2021. url: https://www.cvedetails.com/cve/CVE-2020-10135/

[18]. CVE Details. CVE-2020-35473. 2022. url: https://www.cvedetails.com/cve/CVE-2020-35473/

[19]. Medieval Software. Bluetooth File Transfer. url: https://play.google.com/store/apps/details?id=it.medieval.blueftp&hl=es_419&gl=US

[20]. Modern Perfomance Develop. Transfer archivos Bluetooth url: https://play.google.com/store/apps/details?id=com.modernappdev.btfiletf&hl=es_419&gl=US

[21]. Lucky Developer. Compartir aplicación. url: https://play.google.com/store/apps/details?id=com.luckydeveloper.apkshare&hl=es_419&gl=US

[22]. Bluetooth. About Us. url: https://www.bluetooth.com/about-us/

[23]. Android. Qué es Android. url: https://www.android.com/intl/es_es/what-is-android/

[24]. Kotlin. Get started. url: https://kotlinlang.org/docs/getting-started.html

[25]. Bluetooth. Bluetooth® Core Specification Version 5.4 - Technical Overview. 2023. url: https://www.bluetooth.com/bluetooth-resources/bluetooth-core-specification-version-5-4-technical-overview/

[26]. Android. Android para desarrolladores. url: https://developer.android.com/?hl=es-419

[27]. Android. Cómo obtener el resultado de una actividad. url: https://developer.android.com/training/basics/intents/result?hl=es-419

[28]. Java. Learn Java. url: https://dev.java/learn/

[29]. Android Studio. Android Studio Flamingo. 2022 url: https://developer.android.com/studio/releases?hl=es-419

[30]. Android Studio. Uso avanzado del emulador. url: https://developer.android.com/studio/run/advanced-emulator-usage?hl=es-419#limitations

[31]. Figma. url: https://www.figma.com/

[32]. Git. Documentation. url: https://git-scm.com/doc

[33]. GitHub. url: https://github.com/

[34]. Excalidraw. url: https://docs.excalidraw.com/

[35]. Test de accesibilidad. url: https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor

[36]. MVC. url: https://developer.mozilla.org/es/docs/Glossary/MVC

[37]. Jetpack Compose. url: https://developer.android.com/jetpack/compose?hl=es-419

[38]. Tink, url: https://developers.google.com/tink?hl=es-419

[39]. AesGcmJce url: https://github.com/google/tink/blob/master/java_src/src/main/java/com/google/crypto/tink/subtle/AesGcmJce.java

[40]. Margaret Rouse. Intercambio de claves Diffie-Hellman (intercambio de claves exponencial). 2019. url: https://www.computerweekly.com/es/definicion/Intercambio-de-claves-Diffie-Hellman-intercambio-de-claves-exponencial

[41]. Redacción KeepCoding. ¿Qué es la criptografía de curva elíptica? 2023. url: https://keepcoding.io/blog/que-es-la-criptografia-de-curva-eliptica/

[42]. Java.security. url: https://developer.android.com/reference/java/security/package-summary

[43]. Javax.crypto. url: https://developer.android.com/reference/javax/crypto/package-summary

[44]. W3C. Accessibility Guidelines (WCAG) 3.0. 2021. url: https://www.w3.org/TR/wcag-3.0/#visual-contrast-of-text

[45]. Material. Introduction. url: https://m2.material.io/design/introduction

[46]. NavController. url: https://developer.android.com/reference/androidx/navigation/NavController

[47]. Jetpack. url: https://developer.android.com/jetpack?hl=es-419

[48]. BroadcastReceiver. url: https://developer.android.com/guide/components/broadcasts?hl=es-419

[49]. Hilt. url: https://developer.android.com/training/dependency-injection/hilt-android?hl=es-419